UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

PG477

R_3D_3
Real Robo Rally
Dare-Devil Droids

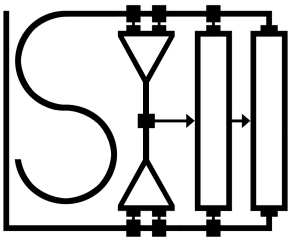
Zwischenbericht

20. April 2006

INTERNE BERICHTE
INTERNAL REPORTS

LS XII (Embedded System Design)
Fachbereich Informatik
Universität Dortmund

Betreuer:
Heiko Falk
Robert Pyka



GERMANY · D-44221 DORTMUND

Zwischenbericht

R3D3

REAL ROBO RALLY
DARE-DEVIL DROIDS

Judith Ackermann, Manuela Engels, Nina Grau, Harald Günther,
Daniel Höcker, Michael Kern, Robert Muhrbeck, Michael Poch,
Daniel Schulte, Jules Souna, Rene Thimel, Benjamin Titz

20. April 2006
Universität Dortmund

Inhaltsverzeichnis

1 Einleitung **13**
- 1.1 Motivation . . . 13
- 1.2 Aufgabenstellung . . . 14
  - 1.2.1 Ziele . . . 14
  - 1.2.2 Minimalziele . . . 16
- 1.3 Realisierung . . . 17
  - 1.3.1 Organisation . . . 17
  - 1.3.2 Teilnehmer . . . 18
  - 1.3.3 Team-Einteilung . . . 18
- 1.4 Erweiterungsmöglichkeiten . . . 19
- 1.5 Gliederung des Dokumentes . . . 19

2 Das Spiel **21**
- 2.1 Das Spiel im Original . . . 21
  - 2.1.1 Über das Spiel . . . 21
  - 2.1.2 Idee des Spiels . . . 21
  - 2.1.3 Vorbereitung . . . 22
  - 2.1.4 Spielablauf . . . 24
  - 2.1.5 Ende des Spiels . . . 27
  - 2.1.6 Varianten . . . 27
  - 2.1.7 Optionskarten . . . 28
  - 2.1.8 Robo Rally 2005 . . . 32
- 2.2 Unsere Realisierung des Spiels . . . 33
  - 2.2.1 Verwendete Varianten . . . 34
  - 2.2.2 Optionskarten . . . 34
  - 2.2.3 Verwendete Neuheiten aus Robo Rally 2005 . . . 34
- 2.3 Das R_3D_3 Design . . . 34

3 Hardware **37**
- 3.1 Gemeinsamkeiten . . . 37
  - 3.1.1 Erstellen von Layouts . . . 37
  - 3.1.2 Ätzen von Platinen . . . 38
  - 3.1.3 Mikrocontroller . . . 40
- 3.2 Kommunikation . . . 54
  - 3.2.1 Host PC - Spielfeldbox . . . 54
  - 3.2.2 Spielfeldbox - Spielfeldbox . . . 55
  - 3.2.3 Spielfeldbox - Roboter . . . 56

3.3 Roboter . . . 59
3.3.1 Anforderungen . . . 59
3.3.2 Standardlösungen . . . 60
3.3.3 Komponentenauswahl . . . 65
3.3.4 Testboard . . . 73
3.3.5 Mechanischer Aufbau . . . 75
3.3.6 Elektronischer Aufbau . . . 80
3.3.7 Programmierung . . . 90
3.3.8 Ergebnisse . . . 112
3.4 Spielfeld . . . 113
3.4.1 Allgemeiner Aufbau . . . 113
3.4.2 Spielfeldbox - Material . . . 115
3.4.3 Unterkonstruktion . . . 116
3.4.4 Oberfläche . . . 118
3.4.5 Verbindungen der Felder . . . 122
3.4.6 Elektronischer Aufbau der Spielfeldes . . . 126
3.4.7 Testboards . . . 128
3.4.8 Gesamtplatine . . . 137
3.4.9 Programmierung . . . 138
3.5 Integrationstests . . . 142
3.5.1 Test der Kommunikation zwischen Roboter und Spielfeld . . . 143
3.5.2 Test der Befehlsausführung im Roboter . . . 145
3.6 Host-Interface . . . 146
3.6.1 Mechanischer Aufbau . . . 146
3.6.2 Elektronik . . . 147
3.6.3 Programmierung . . . 147

4 Software **149**
4.1 Einleitung . . . 149
4.2 Architektur . . . 149
4.2.1 Anforderungen . . . 149
4.2.2 Überblick . . . 150
4.3 Protokoll-Stack . . . 151
4.3.1 Einleitung . . . 151
4.3.2 Routing Protokoll . . . 152
4.3.3 High-Level Protokoll . . . 159
4.3.4 Scheduler . . . 165
4.4 Host-Applikation . . . 166
4.4.1 Bedienkonzept . . . 166
4.4.2 wxWidgets . . . 166
4.4.3 Programmablauf . . . 167
4.4.4 GUI-Elemente . . . 168

5 Zusammenfassung **173**

5.1 Status 173
5.1.1 Hardware 173
5.1.2 Software 174
5.2 Ausblick 174
5.2.1 Hardware 174
5.2.2 Software 175

6 Anhang **177**
6.1 Schaltpläne 177
6.2 Platinenlayouts 184
6.3 Mechanischer Aufbau des Spielfelds 195

Abbildungsverzeichnis

2.1 Ein Spielplan der Version 2005 . . . 22
2.2 Ein Roboter der deutschen Spieleversion . . . 23
2.3 Die Programmierkarten der Version 2005 . . . 24
2.4 Die Roboter der Version 2005 . . . 32
2.5 Docking Bay Board Seite 1 . . . 33
2.6 Docking Bay Board Seite 2 . . . 33

3.1 Keil uVision: Neues Projekt anlegen . . . 49
3.2 Keil uVision: GNU Tools einstellen . . . 50
3.3 Keil uVision: Linker einstellen . . . 50
3.4 Keil uVision: Hex Ausgabe aktivieren . . . 51
3.5 Hauptfenster des LPC2000 Flash Utility . . . 51
3.6 Uploadfenster für den Flashspeicher . . . 52
3.7 Auswahlbildschirm des SAM-BA Flash Tools . . . 53
3.8 Fehlermeldung des SAM-BA Flash Tools . . . 53
3.9 Hauptbildschirm des SAM-BA Flash Tools . . . 54
3.10 Lock-Warnung des SAM-BA Flash Tools . . . 55
3.11 Schema der Kommunikationsverbindungen Box - Box TODO:+DIODEN . 55
3.12 Ergebnisse des Oscillographentests . . . 56
3.13 Schematische Darstellung einer H-Brücke . . . 67
3.14 Anschluss eines Schrittmotors am L293D (Quelle: [B+06a]) . . . 67
3.15 Komponentenübersicht Roboter . . . 73
3.16 Grundriss der Bodenplatte mit Aussparungen in Originalgröße . . . 75
3.17 Die bemaßte Bodenplatte . . . 76
3.18 Die Unterseite der Bodenplatte mit den Filzgleitern . . . 77
3.19 Bodenplatte mit Getrieben, Rädern, Achsen und Winkeln . . . 78
3.20 Der Roboterprototyp (Ansicht hinten) . . . 78
3.21 Der Roboterprototyp (Ansicht links) . . . 79
3.22 Der Roboterprototyp (Ansicht vorn) . . . 79
3.23 Der Roboterprototyp (Ansicht oben) . . . 79
3.24 Platinenaufteilung . . . 81
3.25 Bodenplatine . . . 82
3.26 Nebenplatine . . . 83
3.27 Hauptplatine . . . 85
3.28 Programmablauf Roboter . . . 91
3.29 Datei: main.c Initialisierung . . . 91
3.30 Datei: main.c Akkustand prüfen . . . 92

3.31 Datei: main.c Auftrag . . . 92
3.32 Datei: main.c Nachricht bestätigen . . . 93
3.33 Datei: main.c Nachricht auswerten . . . 94
3.34 Datei: robo.c Akkuüberprüfung . . . 95
3.35 Nachrichtenempfang . . . 96
3.36 Nachrichtenversand . . . 96
3.37 Datei: com.c Realisierung der Zielfrequenz . . . 97
3.38 Motorsteuerung Funktionenübersicht . . . 100
3.39 Motoransteuerungstabelle . . . 101
3.40 Idee für Drehkorrektur (Pseudocode) . . . 106
3.41 $\mathcal{A}$ - Ausgangssituation . . . 107
3.42 $\mathcal{A}$ - Fehlersituation . . . 107
3.43 $\mathcal{A}$ - Folgefehler . . . 107
3.44 $\mathcal{A}$ - Korrekturausrichtung . . . 108
3.45 $\mathcal{A}$ - Nächster Zug . . . 108
3.46 $\mathcal{A}$ - Weitere Korrektur . . . 108
3.47 $\mathcal{A}$ - Letzter Zug . . . 108
3.48 $\mathcal{B}$ - Kleiner Fehler . . . 108
3.49 $\mathcal{B}$ - Dessen Korrektur . . . 108
3.50 $\mathcal{C}$ - Fehlersituation . . . 109
3.51 $\mathcal{C}$ - 1. Maximumssuche . . . 109
3.52 $\mathcal{C}$ - Drehung . . . 109
3.53 $\mathcal{C}$ - 2. Maximumssuche . . . 110
3.54 $\mathcal{C}$ - Inverse Drehung . . . 110
3.55 $\mathcal{C}$ - Drehkorrektur . . . 110
3.56 Der komplette Roboterprototyp . . . 112
3.57 Spielfläche mit Parkplatz . . . 115
3.58 Ansicht der Unterkonstruktion mit Blöcken . . . 117
3.59 Zusammenhalten der Gesamtkonstruktion durch Magnete . . . 117
3.60 Ansicht der Unterkonstruktion . . . 118
3.61 Aufsicht der Unterkonstruktion . . . 119
3.62 Spielfläche mit Laser . . . 119
3.63 Befestigung der Makrolon-Platte . . . 120
3.64 Plätze für die Drucktaster in der Spielfläche . . . 121
3.65 Polung der Magnete in einem Winkel . . . 123
3.66 Polung der Magnete für eine Spielfeldseite . . . 123
3.67 Federbügelkontakt von Elco . . . 124
3.68 Seitenansicht auf eine Spielfeldbox . . . 124
3.69 Stromversorgung über die Winkel . . . 125
3.70 Adressen der Felder und LEDs . . . 127
3.71 Steckerbelegung der Testboards für das Spielfeld . . . 129
3.72 Position der Dioden und Phototransistoren . . . 130
3.73 74HCT259-Baustein . . . 131

3.74 74HCT138-Baustein . . . 131
3.75 Multiplexer . . . 132
3.76 TTL-Platine . . . 133
3.77 Resultierendes PWM-Signal am Pinout . . . 137
3.78 Pointer auf Parameter . . . 139
3.79 Befehlstabelle . . . 140

4.1 Architekturüberblick . . . 150
4.2 Datei: Routing.h . . . 156
4.3 Statemachine . . . 163
4.4 Splashscreen . . . 169
4.5 Hauptbildschirm . . . 169
4.6 Übersichtsbildschirm . . . 170
4.7 Karten . . . 171
4.8 Kombinierte Karten . . . 171
4.9 Optionskarten . . . 172

6.1 Schaltplan Testboard AT91SAM7S32 . . . 177
6.2 Schaltplan Roboter Hauptplatine . . . 178
6.3 Schaltplan Roboter Nebenplatine . . . 179
6.4 Schaltplan Testboard Feld . . . 180
6.5 Schaltplan Testboard CPLD . . . 181
6.6 Schaltplan Feldplatine Version 2.0 . . . 182
6.7 Schaltplan Host-Interface . . . 183
6.8 Platinenlayout Testboard AT91SAM7S32 - Routing - Oberseite . . . 184
6.9 Platinenlayout Testboard AT91SAM7S32 - Routing - Unterseite . . . 184
6.10 Platinenlayout Testboard AT91SAM7S32 - Bauteile - Oberseite . . . 185
6.11 Platinenlayout Testboard AT91SAM7S32 - Bauteile - Unterseite . . . 185
6.12 Platinenlayout Roboter Hauptplatine - Routing - Oberseite . . . 186
6.13 Platinenlayout Roboter Hauptplatine - Routing - Unterseite . . . 186
6.14 Platinenlayout Roboter Nebenplatine Bauteile (nur Oberseite) . . . 187
6.15 Platinenlayout Roboter Nebenplatine Routing Oberseite . . . 187
6.16 Platinenlayout Roboter Nebenplatine Routing Unterseite . . . 187
6.17 Platinenlayout Testboard Feld - Routing - Oberseite . . . 188
6.18 Platinenlayout Testboard Feld - Routing - Unterseite . . . 188
6.19 Platinenlayout Testboard Feld - Bauteile - Oberseite . . . 189
6.20 Platinenlayout Testboard Feld - Bauteile - Unterseite . . . 189
6.21 Platinenlayout Testboard CPLD - Routing - Oberseite . . . 190
6.22 Platinenlayout Testboard CPLD - Routing - Unterseite . . . 190
6.23 Platinenlayout Testboard CPLD - Bauteile - Ober- und Unterseite . . . 191
6.24 Platinenlayout Feldplatine Version 2.0 - Routing - Oberseite . . . 192
6.25 Platinenlayout Feldplatine Version 2.0 - Routing - Unterseite . . . 192
6.26 Platinenlayout Feldplatine Version 2.0 - Bauteile - Oberseite . . . 193
6.27 Platinenlayout Feldplatine Version 2.0 - Bauteile - Unterseite . . . 193

6.28 Platinenlayout Host-Interface - Routing - Oberseite 194
6.29 Platinenlayout Host-Interface - Routing - Unterseite 194
6.30 Platinenlayout Host-Interface - Bauteile - Oberseite 194
6.31 Spielfelfbox von oben gesehen 195
6.32 Spielfeldbox von der Seite gesehen 196

1 Einleitung

In diesem ersten Kapitel möchten wir einen Überblick über die Projektgruppe 477: „R_3D_3 - Real Robo Rally: Dare-Devil Droids“ vermitteln. In diesem Projekt wird das Spielkonzept der „Robo Rally“ Spiele mit Hilfe interagierender Roboter umgesetzt.
Neben der Motivation (Abschnitt 1.1) zum Projekt wird insbesondere die Aufgabenstellung (Abschnitt 1.2) und die Planung der Realisierung (Abschnitt 1.3) beleuchtet. Eine Vorstellung möglicher Projekterweiterungen (Abschnitt 1.4) rundet den Überblick über die Arbeit der Projektgruppe ab.
Im Anschluss wird der Aufbau dieses Zwischenberichtes (Abschnitt 1.5) dargestellt.

1.1 Motivation

Unabhängig vom Alter empfinden viele Menschen Brettspiele als eine angenehme und unterhaltsame Möglichkeit zur Freizeitgestaltung. Die unzähligen Spielideen, die angeboten werden, sind ein starkes Indiz für die anhaltende Popularität der Spielart. Vertreter sind ganz klassische Spiele wie Schach, das schon seit Jahrhunderten Menschen fasziniert, oder das Gesellschaftsspiel Monopoly, bis hin zu den jährlich erscheinenden Variationen der „Mensch ärger dich nicht“ Spielidee.

Einer gelungenen Idee folgt auch die „Robo Rally“ Spielserie [AMI99][AMI00]. Die Grundidee besteht darin, mit Spielfiguren, die als Roboter gestaltet sind, auf einer Spielfläche eine Folge von Zielpunkten anzusteuern.
Die Spielfiguren können mit Anweisungskarten - von den Spielern zu „Programmen“ zusammengesetzt - gesteuert werden. Diverse Elemente der Spielfelder wirken auf die Roboter-Spielfiguren ein - so gibt es Transportbänder, Mauern und verschiedene schädigende Elemente. Die Roboter können sich auch gegenseitig beschädigen, wodurch jeweils die Fähigkeit zur Programmausführung beeinträchtigt wird.
Durch die Vielzahl der aktiven Spielfeldelemente und die unvorhersehbare Reihenfolge, in der Anweisungskarten gezogen werden, bietet das Spiel durch alle Runden hinweg viele Überraschungsmomente, wodurch den Spielern die Fähigkeit zur ständigen Anpassung ihrer Strategie und der fortwährenden Abschätzung der Aktionen anderer Spieler abverlangt wird. Diese begrenzte Unvorhersagbarkeit des Spielgeschehens und die Freude über eine richtig abgeschätzte Ausführung der Programme (aber auch Schadenfreude bei Missgeschicken anderer Mitspieler) sind sicherlich die Hauptgründe für eine heitere und angenehme Spielatmosphäre.

Gegenüber der klassischen Implementierung der Spielidee, bestehend aus einem Papierspielbrett und den üblichen Kunststofffiguren, gibt es die Möglichkeit, das Spiel als eine Softwarelösung umzusetzen.

Obwohl dies die nahe liegendste Lösung wäre, erfüllt sie doch schlicht einige Anforderungen und Wünsche nicht. Das markanteste Defizit ist sicherlich die fehlende haptische Komponente. Ferner sollte bei der Wahl der Implementierungsart die Überlegung, dass Menschen seit jeher durch den Anschein des Spiels gegen eine Maschine fasziniert wurden, nicht außer Acht gelassen werden.
Stellvertretend sei hier Maelzels Schachspieler von 1783 [Poe36] - ein von einem Menschen gesteuerter Schach spielender „Automat“ - genannt, der das Publikum bereits zu Zeiten der Aufklärung fasziniert hat.

Die neuartige Umsetzung der Robo Rally Spielidee durch die Projektgruppe soll daher auf realen, selbst fahrenden Roboter-Spielfiguren basieren, die sich selbständig auf einer 12x12 Spielfelder großen Fläche bewegen.
Die Interaktion mit dem Benutzer kann durch Steuerkonsolen erfolgen, auf denen die Anweisungssequenzen mit virtuellen oder echten Spielkarten erstellt werden.
Mit einer modularen Spielfläche wird gegenüber dem Originalspiel ein weiterer Vorteil erzielt - nämlich, dass durch Variation der Spielfläche unterschiedliche Schwierigkeitsstufen vorgegeben werden können.
Schließlich können Roboter auch selbständig agieren, um beispielsweise fehlende Mitspieler zu ersetzen.

1.2 Aufgabenstellung

Die Aufgabenstellung der Projektgruppe besteht aus zwei Teilen:

- Aus der Motivation heraus sind allgemeine Ziele zu entwickeln. Diese Ziele sollen durch die Projektgruppe erreicht werden und werden in Abschnitt 1.2.1 vorgestellt.
- Für die genaue Aufgabenstellung sind diese Ziele jedoch zu konkretisieren. Es werden daher aus den allgemeinen Zielen Minimalziele extrahiert, die das Projektteam verbindlich zu erfüllen hat. Diese werden in Abschnitt 1.2.2 dargestellt.

Entsprechend dieser Aufteilung enthalten die Zielformulierungen in Abschnitt 1.2.1 eine Reihe von Lösungsansätzen und Ideen, deren Umsetzung nicht verpflichtend ist. Ferner werden auch schon Erweiterungsmöglichkeiten - wie sie in Abschnitt 1.4 genau vorgestellt werden - aufgegriffen.

1.2.1 Ziele

Im Rahmen dieser Projektgruppe wird ein Spiel erstellt, das die Spielidee der Robo Rally Spiele umsetzt. Die Kerneigenschaften dieser Implementierung sind eine modulare, aktive Spielfläche, auf der selbst fahrende Roboter-Spielfiguren das Spielgeschehen ausführen.
Die Aufgabenstellung ist dementsprechend in zwei Hauptaufgabenbereiche zu gliedern. Zum einen sind Spielfiguren zu entwickeln, zum anderen die Spielfläche.

1.2.1.1 Spielfiguren

Die Spielfiguren sollen höchstens $4,5\ cm$ Durchmesser und eine Höhe von etwa $6\ cm$ haben.

Die mechanische Realisierung der Spielfiguren besteht in der Entwicklung einer gemeinsamen Plattform für alle - mindestens vier - Spielfiguren. Insbesondere geht es dabei um die Entwicklung eines Antriebs. Der mechanische Aufbau muss ferner hinreichend Platz und entsprechende Haltevorrichtungen für Akkus und Steuerplatine vorsehen. Ebenfalls Bestandteil des mechanischen Aufbaus ist die Anordnung der Sensoren für die Kommunikation und Positionsfindung.

Der elektrotechnische Aufbau der Spielfiguren wird aus einer Mikrocontrollerschaltung bestehen, die für die Ansteuerung der Motoren und die Anbindung der Datenübertragungseinrichtung notwendig ist. Eine Schaltung zum Wiederaufladen der Akkus wird nicht in die Spielfiguren integriert, sondern in Form einer Ladestation realisiert.
Für die Datenübertragung zwischen dem Spielfeld und den Spielfiguren gibt es zahlreiche Ansätze. Zunächst soll einer optischen Kommunikation der Vorzug gegeben werden.

Die Auswahl des Mikrocontrollers ist ein kritischer Aspekt der Konzeptphase dieser Projektgruppe (PG). Es gibt hierbei zahlreiche gegenläufige Anforderungen, zwischen denen ein sinnvoller Kompromiss gefunden werden muss. So soll die Hardware ausreichend Rechenleistung und Speicher bieten, möglichst wenig Energie verbrauchen, leicht zu programmieren sein, eine gute Unterstützung durch Entwicklungstools haben und speziell in dieser PG auch so gewählt werden, dass der gleiche Mikrocontrollertyp für die Spielfeldkomponenten verwendet werden kann.

Die Implementierung der Software der Spielfiguren beinhaltet zeitkritische hardwarenahe Komponenten - beispielsweise die Ansteuerung von Motoren oder die Auswertung von Sensoren. Darüber hinaus muss ein Protokollstack für die Kommunikation mit einem Spielfeld und anderen Spielfiguren implementiert werden.
Schließlich werden die Spielfiguren eine gewisse abstrakte Steuerlogik besitzen, die sie auf Anforderungen reagieren lässt. Beispielsweise ist hier die Verarbeitung von Kommandos wie „Fahre ein Feld vor“ und ähnlichen notwendig.

Die Mikrocontroller werden ohne jegliches Betriebssystem ausgeliefert, so dass eine strukturierte Implementierung der Software Aspekte der Betriebssystementwicklung aufweisen wird. Für die Navigation wird auf Methoden aus der Robotik zurückgegriffen und für die Kommunikation muss ein Protokollstack entwickelt werden. Eine zuverlässige und flexible Implementierung wird durch den Gebrauch von Methoden aus dem Bereich des Protocol Engineering geprägt sein.

1.2.1.2 Spielfläche

Ähnlich gestalten sich die Aufgaben bei der Entwicklung der Spielfläche. Die Spielfläche soll aus Blöcken zu jeweils 3x3 Spielfeldern zusammengesetzt werden. Es werden mindestens 16 derartiger Blöcke benötigt, um eine dem Originalspiel entsprechende Spielfläche von 12x12 Feldern zu bieten. Ein Spielfeld soll eine Kantenlänge von 5 *cm* aufweisen.

Der mechanische Aufbau der Spielfläche ist jedoch wesentlich einfacher als der der Spielfiguren, da keine beweglichen Teile benötigt werden. Jedoch muss ein Block an vier Seiten Kontakte für Stromversorgung und Kommunikation bereitstellen.
Für die Kommunikation soll ein serielles Protokoll verwendet werden, so dass die Anzahl der Kontakte minimiert werden kann. Die Anordnung der Kontakte ist so zu wählen, dass ein beliebiges Zusammenstecken der Blöcke möglich ist, und dabei die Oberseiten der Blöcke eine durchgehende Spielfläche ergeben.
Dabei besteht ein Spielfeldblock aus 9 Spielfeldern, die - für die Mitspieler erkennbar - entsprechend den Spielfeldeigenschaften gestaltet werden. Ferner sind in der Oberseite lichtdurchlässige Bereiche vorzusehen, um ggf. eine lichtbasierende Kommunikation mit den Robotern integrieren zu können.

Der elektrotechnische Aufbau der Spielfeld-Blöcke besteht im Wesentlichen aus dem Mikrocontroller, der nahezu ohne weitere Komponenten direkt die Ansteuerung der Verbindungen zu den anderen Blöcken und die der Oberflächenelemente übernimmt. Hierzu soll der gleiche Mikrocontrollertyp wie in den Spielfiguren verwendet werden.

Die Software der Spielfeld-Blöcke hat mehrere Teilaufgaben zu erledigen.
In einer ersten Phase müssen die Blöcke eine Selbstorganisation durchführen. Da die Spieler die Möglichkeit haben sollen, die Blöcke beliebig anordnen zu können, muss das Kommunikationsprotokoll zwischen den Blöcken die Erstellung eines konsistenten Modells der Spielfläche ermöglichen. Für den Spielbetrieb müssen Methoden zum Routing von Datenpaketen zwischen den Blöcken bereitgestellt werden. Ferner wird eine Softwarekomponente benötigt, die den Datenaustausch mit den Spielfiguren durchführen kann. Und schließlich muss ein Spielfeld entsprechend den Spielregeln Aktionen anstoßen können. Insbesondere muss es in der Lage sein, in Kooperation mit anderen Blöcken die Ausführung der Spiel-Programmschritte zu ermöglichen.

Hierbei werden im hohen Umfang Aufgaben aus dem Bereich der Verteilten Systeme bearbeitet werden. Eine besondere Eigenschaft dieses Verteilten Systems wird die reguläre Anordnung der Blöcke sein. Sie ergibt ein Mesh-Netzwerk, so dass beispielsweise auf Konzepte zu effizienten Routingverfahren in parallelen Computer-Architekturen zurückgegriffen werden kann.

1.2.2 Minimalziele

Konkret sind folgende Ziele von der Projektgruppe zu erreichen:

1. Kombinierter Entwurf der Roboter- und Spielflächen-Hardware inklusive Auswahl der zu verwendenden Bauelemente (Controller, Antrieb) für Roboter und Spielfläche.

   Der Hardware-Entwurf ist so vorzunehmen, dass sich ein Roboter unter Nutzung von Navigationshilfen der Spielfläche (optische oder induktive Markierungen) zielsicher von einem Spielfeld zu einem anderen bewegen und mit dem Spielfeld kommunizieren kann.

2. Definition und Entwurf der in der Spielfläche zu verwendenden Netzwerk-Protokolle.

   Die in der Spielfläche eingesetzte Software muss in der Lage sein, eine gezielte Kommunikation zwischen benachbarten Spielfeldern, zwischen Roboter und Spielfeld, Roboter und Roboter und zwischen Host-Computer und Roboter zu unterstützen. Die Software muss weiterhin fähig sein, sich selbständig auf Veränderungen der Spielfläche durch individuelles Zusammenstecken von Spielfeld-Blöcken vor Spielbeginn einzustellen.

3. Implementierung einer Steuerungssoftware zur Kommunikation mit Robotern und Spielfläche von einem Desktop-Computer aus (Host-Interface).

   Mit Hilfe dieser Software sollen die Roboter den Spielregen entsprechend gesteuert werden können.

4. Demonstration der Funktionsfähigkeit der erstellten R_3D_3 Hard- und Software im Spielbetrieb.

1.3 Realisierung

Ein Überblick über die Organisation und Vorgehensweise in der Projektgruppe mag das Verständnis über Art und Umfang der dargestellten Arbeiten schärfen.

1.3.1 Organisation

Die Projektgruppe R_3D_3 findet im Wintersemester 2005/2006 und im Sommersemester 2006 jeweils mit acht Semesterstundenwochen für zwölf Studenten an der Universität Dortmund statt. Mit diesem Zwischenbericht endet das erste Halbjahr.
Betreut wird das Projekt von Dr. Heiko Falk und Dipl.-Inform. Robert Pyka vom Lehrstuhl 12 für Technische Informatik und Eingebettete Systeme der Fakultät Informatik.
Der Internetauftritt der Projektgruppe ist unter der URL

`http://ls12-www.cs.uni-dortmund.de/r3d3/`

zu finden.

1.3.2 Teilnehmer

Die zwölf studentischen Teilnehmer verfolgen in vier Teams die Ziele der Projektgruppe (vgl. 1.3.3):

- Team A: Judith Ackermann, Nina Grau, Jules Souna, Rene Thimel;
- Team B: Manuela Engels, Harald Günther, Daniel Schulte, Benjamin Titz;
- Team C: Daniel Höcker, Michael Poch;
- Team D: Michael Kern, Robert Muhrbeck.

1.3.3 Team-Einteilung

Schon in den Zielbeschreibungen in 1.2.1 ist deutlich geworden, dass die Arbeiten in der Projektgruppe mehrere Schwerpunkte umfassen. Neben den offensichtlichen Schwerpunkten Spielfelder sowie Spielfiguren sind Netzwerkprotokolle, aber auch Hostsoftware und -Interface zu bearbeiten.
Analog zu diesen vier Schwerpunkten wird eine Teamstruktur geschaffen.

In den Teams sind die entsprechenden Probleme zu bearbeiten und Lösungen zu finden. Dabei erfordern enge Verknüpfungen der Arbeitsbereiche eine gute Absprache und hohe Kooperation zwischen den Teams, unterschiedliche Arbeitsbelastungen aber auch teamübergreifende Probleme zudem eine gegenseitige Unterstützung.

Dabei gliedert sich die Aufgabenteilung zwischen den Teams wie folgt:

- Team A ist für die Hardware der Spielfläche zuständig.

  In der Planungsphase ist festzulegen, wie die Schnittstellen zu benachbarten Spielfeld-Blöcken aussehen werden. Weiterhin muss das Interface zur Navigation und Kommunikation mit den Robotern definiert werden.
  Zunächst wird ein geeigneter Mikrocontroller für einen Spielfeld-Block ausgewählt und ein Prototyp eines Spielfeld-Blocks entwickelt, gefertigt und getestet. Nach erfolgreichem Funktionstest des Prototyps und bestandenem Integrationstest mit einem Roboter müssen weitere Spielfeld-Blöcke gefertigt werden, um eine komplette bespielbare Spielfläche zu erhalten.

- Team B beschäftigt sich mit der Konstruktion der Roboter.

  Ähnlich dem Vorgehen im Team A wird zuerst ein Prototyp entwickelt und getestet. Anschließend ist für einen vollständigen Spielbetrieb die Mindestanzahl von vier funktionsfähigen Robotern herzustellen.

- Team C beschäftigt sich im Wesentlichen mit der Planung und Realisierung der Netzwerk-Protokolle und -Software.

Hierzu zählt unter anderem auch die Inbetriebnahme der Compiler- und Simulator-Umgebungen für die auszuwählende Mikrocontroller-Plattform.

- Der Schwerpunkt von Team D liegt in der Entwicklung der Software für das Host-Interface des Spiels.

  Basierend auf einem Linux-PC ist eine geeignete Verbindung (z.B. via RS 232) zur Spielfläche herzustellen. Anhand des von Team A konstruierten Prototypen eines Spielfeld-Blocks ist zu zeigen, dass eine Programmierung der Spielfeld-Blöcke erfolgreich vom Host-PC durchgeführt werden kann.
  Eine weitere Aufgabe dieses Teams besteht im Aufbau einer graphischen Benutzeroberfläche, um den Spielern am Host-Rechner die Zuteilung von Programmierkarten etc. zu ermöglichen, mit denen die Spieler ihre Roboter auf der Spielfläche ansteuern können.

1.4 Erweiterungsmöglichkeiten

Sofern der zeitliche Rahmen dies zulässt, wären folgende Optionen zur Erweiterung des bis dahin realisierten R_3D_3 Spiels denkbar:

- Erweiterung des R_3D_3 Spiels um Regeln und Spiel-Varianten, die während der Konzeptphase aus Gründen des hohen Aufwandes ausgeschlossen wurden.
- Entwicklung externer Spiel-Konsolen.

  Diese sollen die Interaktion der Spielfläche mit dem Host-PC während des Spielbetriebs ablösen. Anstelle eines gemeinsamen PC erhält jeder Spieler eine dedizierte Konsole, die an beliebiger Stelle am Spielflächenrand angeschlossen wird. Die Steuerbefehle eines Spielers für seinen eigenen Roboter werden dann von der Konsole an die Spielfläche übertragen.

1.5 Gliederung des Dokumentes

Eine Gliederung des Zwischenberichtes analog zu den Teameinteilungen (vgl. 1.3.3) erscheint nahe liegend. Da jedoch an einigen Stellen teamübergreifende Problemstellungen zu lösen sind und sich ferner Arbeitsbereiche der Teams überschneiden, wird von dieser Struktur abgewichen. Lediglich die grobe Aufteilung in ein Hardware- und ein Softwarekapitel bleiben bestehen.

Einführend wird zuvor das Spiel Robo Rally in Kapitel 2 vorgestellt und die vom R_3D_3 Team angestrebte Umsetzung näher erläutert, denn der Realisierungsform entsprechend war eine sinnvolle Auswahl an Zusatzfeatures zu treffen.

Im Kapitel 3 wird die Hardware beschrieben. Zunächst werden die die Spielfläche und Spielfiguren gemeinsam betreffenden Aspekte vorgestellt. Da insbesondere die Kommu-

nikation umfangreich und aufwendig ist und zudem alle Hardwarekomponenten betrifft, wird auf sie gesondert eingegangen.
Die individuellen Problemstellungen und Lösungen der Spielfiguren und Spielfeldboxen werden in eigenen Unterkapiteln präsentiert. Es folgt eine Beschreibung der Integrationstests für beide Hardwarekomponenten. Die Beschreibung des Interfaces zum Host beschließt das Kapitel.

Die Darstellung der Software im Kapitel 4 gliedert sich in drei Abschnitte. Bevor der Protokollstack auf der einen Seite und die Host-Applikation auf der anderen Seite näher betrachtet werden, wird ein Überblick über die Softwarearchitektur gegeben, um die einzelnen Komponenten im richtigen Zusammenhang sehen zu können.

Eine Zusammenfassung in Kapitel 5 bestehend aus einem Überblick über den aktuellen Status des Projektes und einen Ausblick auf das zweite Halbjahr der Projektarbeit beschließt diesen Zwischenbericht.

2 Das Spiel

2.1 Das Spiel im Original

In diesem Kapitel soll auf das Spiel „Robo Rally“, seine Geschichte und seine Regeln eingegangen werden, das als Ausgangsbasis für diese Projektgruppe dient. Die Regeln der deutschen Erstausführung sind der Anleitung zu dem Spiel von 1999 [AMI99] und der Erweiterung „Crash and Burn“ [AMI00] entnommen.

2.1.1 Über das Spiel

Das Spiel „Robo Rally“, nach der Idee von Richard Garfield, kann bereits auf eine lange Geschichte zurückblicken. Die erste Veröffentlichung im Jahre 1994 und die letzte im Jahre 2005 zeugen davon, dass seine Idee bis heute lebendig geblieben ist.

In den Vereinigten Staaten von Amerika erschien „Robo Rally“ im Jahre 1994 im Verlag Wizards of the Coast, und 1999 kam es dann auch nach Deutschland, vertrieben von der Amigo Spiel und Freizeit GmbH. Bald gab es Erweiterungen für das Basisspiel. Die Erste, „Crash and Burn“ genannt, wurde in den USA 1997 und im Jahr 2000 dann auch in Deutschland veröffentlicht, allerdings in einer etwas veränderten Variante. Kurz darauf waren dann in englischer Sprache die Spielsätze „Armed and Dangerous“, „Grand Prix“ und „Radioactive“ erhältlich. Da das Spiel in Deutschland nicht den gewünschten Absatz hatte, gab es die Sätze ausschließlich für den amerikanischen und englischen Markt. Da die gesamte Produktion des Spiels irgendwann eingestellt wurde, ist es heute schwerer denn je, noch Spiele der alten Generation zu bekommen. Im Jahr 2005 brachte Wizards of the Coast eine Neuauflage des Spiels „Robo Rally“ heraus. Die Version 2005 hat ein neues Design und bringt ein paar Veränderungen mit sich. Allerdings ist bisher nur eine englische Version erhältlich.

Das Spiel, inklusive aller Erweiterungen, ist für bis zu 8 Spieler ab einem Alter von 12 Jahren geeignet. Die Spieldauer beträgt laut Angaben der Hersteller mindestens 60 Minuten. Erfahrungen zeigen, dass ein Spiel je nach Aufbau der Spielpläne einige Stunden dauern kann.

2.1.2 Idee des Spiels

Bei „Robo Rally“ machen Roboter auf einem Fabrikgelände ein Wettrennen. Jeder Spieler steuert einen Roboter und versucht, diesen als Ersten über die Rennstrecke zu bringen.

Diese wird durch Fabrikelemente, wie z.B. Wände, Laser und Förderbänder und auch durch die Roboter selbst, beeinflusst. Durch Optionskarten, die in den Erweiterungen enthalten sind, können Roboter besondere Eigenschaften erhalten. Da die Roboter sich nahezu zeitgleich bewegen, wird so manches lustiges Chaos verursacht. Wer als Erster alle Checkpoints in der korrekten Reihenfolge passiert hat und das Ziel erreicht, ist der Gewinner des Spiels.

2.1.3 Vorbereitung

Als Grundlage für die folgenden Beschreibungen dienen das deutsche Grundspiel von Amigo (1999) und die passende Erweiterung „Crash and Burn“. Auf die amerikanischen Varianten und insbesondere auf die Version 2005 wird in Abschnitt 2.1.8 eingegangen.

Spielplan

Zu Beginn wird der Spielplan ausgelegt, wobei die Spieler entscheiden können, wie viele und welche der vorhandenen Spielpläne sie benutzen möchten. Spielt man mit mehreren Plänen, so legt man diese zu einem großen Spielplan zusammen, so dass sie genau Seite an Seite aneinander passen. Die Spieler können frei wählen, wie sie diese Pläne zusammenlegen.

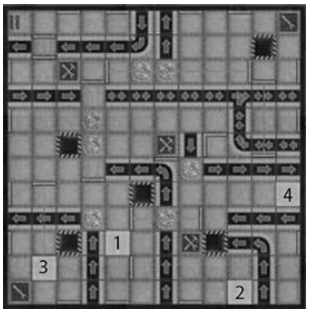


Abbildung 2.1: Ein Spielplan der Version 2005

Rennstrecke festlegen

Als nächstes legt man die Rennstrecke mit Hilfe von Checkpoints, das sind Teilziele, die von den Robotern erreicht werden müssen, fest. Sie sind mit Nummern versehen und müssen in aufsteigender Reihenfolge passiert werden. Beim Checkpoint 1 wird gestartet, Ziel ist der Checkpoint mit der höchsten Nummer. Wie viele Checkpoints verwendet werden und auf welche Felder diese auf dem Spielplan gelegt werden, entscheiden die Spieler. Mögliche Checkpoint-Felder sind jedoch nur leere Felder, Felder mit einer Mauer, Felder mit zwei Mauern, Reparaturfelder mit zwei Schraubenschlüsseln ohne Mauer,

Reparaturfelder mit zwei Schraubenschlüsseln und zwei Mauern, Reparaturfelder mit einem Schraubenschlüssel ohne Mauer, Reparaturfelder mit einem Schraubenschlüssel und einer Mauer.

Material für den Spieler

Sind Spielplan und Rennstrecke festgelegt, werden Programmierkarten, Schadenspunkte, Optionskarten, und für jeden Spieler ein Roboter (wie in Abb. 2.2), eine Programmierhilfe, eine Sicherheitskopie, Lebenspunkte und eine Übersichtstafel bereitgelegt. Programmierkarten sollen die Programme für die Roboter festlegen. Da Roboter während des Spiels beschädigt werden können, gibt es Schadenspunkte, die dies anzeigen. Der Roboter selbst ist eine Plastikfigur, bei der durch einen farbigen Pfeil die Vorder- und Rückseite kenntlich gemacht wird. (In der amerikanischen Urversion waren die Roboter Zinnfiguren, die selbst bemalt werden konnten.) Die Programmierhilfe ist ein Pappkärtchen in Form dieses farbigen Pfeils, auf dem der jeweilige Roboter abgebildet ist. Diese soll gerade Spielanfängern das Programmieren des Roboters erleichtern. Jeder Spieler bekommt außerdem eine Sicherheitskopie seines Roboters, ein quadratisches Pappkärtchen in der Farbe und mit dem Bild des jeweiligen Roboters, der von dem Spieler bewegt wird. Diese wird während des Spiels auf Checkpoints oder Reparaturfelder gelegt, so dass, wenn der Roboter zerstört wird, nach Abgabe eines Lebenspunktes, ein Duplikat des Roboters ab dem Standort der Sicherheitskopie das Spiel fortsetzen kann. Auf der Übersichtstafel werden alle Fabrikelemente und der Programmablauf erläutert. Die Lebenspunkte werden zu Beginn wie folgt verteilt: Wenn zwei bis vier Spieler auf ein bis vier Plänen spielen, erhält jeder Spieler drei Lebenspunkte. Jeder Spieler erhält einen weiteren Lebenspunkt, wenn mit fünf oder mehr Spielern gespielt wird, und einen weiteren Lebenspunkt, wenn mit fünf oder mehr Spielplänen gespielt wird. Es werden also bis zu fünf Lebenspunkte an alle Spieler ausgegeben. In der Erweiterung sind 10 Fabrikelemente als Plättchen mitgeliefert, die auch noch auf leeren Feldern des Spielfeldes verteilt werden können.

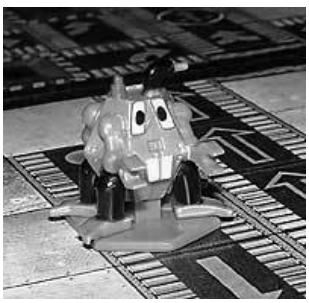


Abbildung 2.2: Ein Roboter der deutschen Spieleversion

2.1.4 Spielablauf

„RoboRally“ wird in Runden gespielt. Dabei besteht jede Runde aus vier Schritten: Dem Austeilen der Programmierkarten, dem Programmieren der Roboter, dem Programmablauf und dem Abschluss der Runde.

Austeilen der Programmierkarten

Alle 84 Karten werden gemischt, und jeder Spieler erhält verdeckt neun Karten auf die Hand. Die Karten zeigen unterschiedliche Befehle, durch die die Roboter gesteuert werden können: Die Vorwärtsbewegung um ein, zwei oder drei Felder, die Rückwärtsbewegung um ein Feld, die Vierteldrehung nach rechts oder links und die halbe Drehung. Erhält ein Spieler in der ersten Runde nur Drehbefehle, so erhält er neun neue Karten. Ab der zweiten Runde verteilt der Kartengeber nur so viele Karten an die Spieler, wie der Schaden ihres Roboters es zulässt. Auf jeder Programmierkarte steht eine Programmnummer. Diese gibt die Reihenfolge an, in der die Spieler ihre Roboter bewegen dürfen. Ein Roboter, dessen Programmkarte die Nummer 840 hat, darf sich vor einem Roboter mit der Programmkartennummer 260 bewegen (vgl. Abb. 2.3).

Abbildung 2.3: Die Programmierkarten der Version 2005

Programmieren der Roboter

Jeder Spieler muss, nachdem die Karten alle verteilt worden sind, ein Programm für seinen Roboter zusammenstellen, das den Roboter steuern soll. Er wählt aus seinen neun Karten fünf aus und legt sie verdeckt so vor sich hin, dass die erste Karte des Programms links und die letzte rechts liegt. Die erste Karte stellt den ersten Befehl dar, und die letzte den fünften Befehl. Nachdem ein Spieler seine Programmierung abgeschlossen hat, legt er die restlichen Karten auf den Stapel der nicht ausgeteilten Karten zurück und seine Programmierhilfe auf den ersten Befehl. Danach darf das Programm nicht mehr verändert werden. In der ersten Runde müssen die ersten Befehle aller Spieler eine Vorwärts- oder Rückwärtsbewegung sein.

Ablauf des Programms

Der Programmablauf sieht vor, dass alle Spieler ihre erste Karte aufdecken, derjenige Roboter, dessen Programmnummer die höchste ist, beginnen darf, und dann alle anderen in absteigender Reihenfolge ihren Befehl ausführen dürfen. Danach treten alle Fabrikelemente in Aktion. Erst dann drehen alle ihre zweite Programmkarte um, und verfahren wie im ersten Schritt. Nachdem alle Fabrikelemente nach Ausführung der letzten Befehle in Aktion getreten sind, werden alle Programmkarten neu gemischt und neu verteilt.

Befehle ausführen Beim Ausführen der Befehle können folgende Ereignisse eintreten: Ein Roboter schiebt einen anderen Roboter, stößt gegen eine Wand, fällt in eine Grube, bewegt sich vom Spielplan, bewegt sich auf einer Öllache, bewegt sich auf ein Portal, bewegt sich in einen Brenner hinein oder bewegt sich durch eine Schwingtür:

- Da immer nur ein Roboter auf einem Feld stehen darf, schiebt ein Roboter, der auf ein Feld ziehen will, das bereits von einem anderen Roboter besetzt ist, diesen vor sich her, unabhängig von der Richtung, in die der andere Roboter gerade zeigt.
- Stößt ein Roboter gegen eine Wand, so wird seine Bewegung blockiert und er kommt vor der Wand zum Stehen. Selbst wenn der Befehl, den der Roboter gerade ausführt, ihn ein Feld weiter ziehen lassen würde, wird sein Zug an der Stelle abgebrochen und er bleibt stehen. Das gilt ebenso für Rückwärtsbewegungen. Dies gilt auch beim Verschieben eines anderen Roboters.
- Fällt ein Roboter in eine Grube, so wird dieser vollständig zerstört. Ein Roboter kann nicht über eine Grube hinweg ziehen.
- Wenn sich ein Roboter über den Spielfeldrand hinaus bewegt, an dem kein anderes Spielfeldbrett anliegt, dann wird dieser ebenfalls vollständig zerstört.
- Eine Öllache verursacht keinen Schaden. Wenn ein Roboter seine Bewegung auf einer Öllache beendet, so rutscht er solange in seiner Bewegungsrichtung weiter, bis er auf einem Feld ohne Öl steht oder von einer Wand oder einem anderen Roboter blockiert wird. Wenn sich zwei Roboter auf derselben Öllache befinden, so kann ein Roboter einen anderen schieben. Sie rutschen dann so lange, bis der vordere Roboter auf einem Feld ohne Öl zum Stehen kommt. Wichtig ist, dass ein Roboter erst dann rutscht, wenn er seine Bewegung beendet, nicht solange er noch in Bewegung ist. Startet ein Roboter seine Bewegung auf einem Feld mit einer Öllache, so bewegt er sich ein Feld weniger weit als auf der Programmierkarte angegeben. Wenn der Befehl eine Vorwärtsbewegung um ein Feld vorsieht, so bleibt der Roboter auf der Stelle stehen. Drehbefehle werden von Öllachen nicht beeinflusst.
- Bewegt sich ein Roboter auf ein Portal, so wird der Roboter zu dem gleichfarbigen anderen Portal transportiert. Sollte dieses jedoch besetzt sein, so ist das Portalfeld ausgeschaltet und wird als ein Feld ohne Funktion behandelt. Der Roboter zieht dann darüber hinweg oder bleibt darauf stehen.

- Ein Brenner kann einem Roboter einen Schaden zufügen sowohl dann, wenn dieser sich auf ein Feld mit einem aktiven Brenner bewegt, als auch wenn er sich über ein solches Feld hinweg bewegt. Die Programmschritte, in denen der Brenner aktiv ist, sind auf dem Feld durch Zahlen angegeben.
- Auch Schwingtüren sind nur in bestimmten Programmschritten aktiv, die durch Zahlen angezeigt werden. Ist eine Schwingtür aktiv, so kann sich ein Roboter durch sie hindurch bewegen. Ist sie nicht aktiv, so verhält sie sich wie eine Wand, die Bewegung des Roboters wird dann an dieser Stelle unterbrochen.

Fabrikelemente treten in Aktion Es gibt folgende Fabrikelemente, die in dieser Phase in Aktion treten: Expressbänder, Förderbänder, Schieber, Zahnräder, Pressen, Laser, Brenner, Checkpoints und Reparaturfelder. Wenn die Fabrikelemente in Aktion treten, dann ist diese Reihenfolge einzuhalten:

- Expressbänder transportieren Roboter ein Feld weiter
- Expressbänder transportieren Roboter ein zweites Feld weiter; Förderbänder transportieren Roboter ein Feld weiter
- Schieber, die während des Programmschrittes aktiv sind, schieben Roboter auf ein benachbartes Feld
- Zahnräder drehen Roboter um 90° nach links oder rechts
- Pressen, die während des Programmschrittes aktiv sind, zerstören Roboter
- Laser fügen Roboter Schaden zu
- Brenner, die während eines Programmschrittes aktiv sind, fügen Robotern Schäden zu
- Checkpoints und Reparaturfelder erhalten die Sicherheitskopie eines Roboters, wenn dieser darüber hinweg gezogen ist oder darauf zum Stehen gekommen ist.

Expressbänder transportieren Roboter als erstes ein Feld in Pfeilrichtung, im zweiten Schritt dann erneut ein Feld weiter, wenn der Roboter noch immer auf einem Expressband steht. Gleichzeitig mit dem zweiten Schritt werden Roboter von Förderbändern ein Feld weiter transportiert. Auf den oben erwähnten Fabrikelementen können Ziffern abgebildet sein, die kenntlich machen, in welchem Programmschritt sie aktiv sind. Steht ein Roboter zum Beispiel im zweiten Programmschritt auf einer Presse, die nur im ersten, dritten und fünften Schritt aktiv ist, so passiert ihm nichts!

Abschluss der Runde

Wenn alle Programmschritte ausgeführt worden sind, dann findet der Abschluss der Runde statt. Roboter können ihre Schäden reparieren lassen, wenn sie auf einem Checkpoint

oder einem Reparaturfeld stehen. Auf Reparaturfeldern mit einem Schraubenschlüssel und auf einem Checkpoint kann ein Schadenspunkt abgegeben werden. Ein Spieler kann bis zu zwei Schadenspunkte abgeben, wenn er auf einem Reparaturfeld mit zwei Schraubenschlüsseln steht. Wenn zum Beispiel Karten durch Schadenspunkte blockiert sind, so kann er sich entscheiden, welche der blockierten Karten er wieder frei geben will. Steht der Roboter eines Spielers am Ende der Runde auf einem Reparaturfeld mit zwei Schraubenschlüsseln, so kann er wählen, ob er einen Schadenspunkt abgeben oder eine Optionskarte erhalten will. Dabei ist zu beachten, dass Spieler nur maximal drei Optionskarten besitzen dürfen. Erhalten sie eine vierte Karte, so müssen sie sich sofort entscheiden, welche der Karten sie zurückgeben. Wird ein Roboter zerstört, so muss der Besitzer eine Optionskarte abgeben.

Des Weiteren haben Roboter, die nun auf einem Checkpoint stehen oder während der Runde an einem vorüber gezogen sind, ein Teilziel der Robo Rally erreicht.

Ist ein Roboter während der Runde zerstört worden, so kann der Spieler durch Abgabe eines Lebenspunktes bewirken, dass er von dem Feld, auf dem seine Sicherheitskopie liegt, wieder mit einem Roboterduplikat ins Rennen gehen darf.

2.1.5 Ende des Spiels

Das Spiel „Robo Rally“ ist beendet, wenn ein Roboter, egal in welchem Programmschritt, den letzten Checkpoint überquert oder auf ihm zu stehen kommt. Falls alle Roboter zerstört werden, bevor der letzte Checkpoint erreicht werden konnte, so gewinnt der Spieler, der mit seinem Roboter die meisten Teilziele erreicht hat.

2.1.6 Varianten

Die Roboter besitzen Laser

Im Grundspiel sind noch einige Varianten vorgesehen. Zum Beispiel können Roboter an ihrer Vorderseite Laser besitzen. Diese schießen im Spiel zeitgleich mit den Lasern der Spielpläne. Auch hier können nur blockierende Wände und andere Roboter verhindern, dass man einen Schaden durch den Laser eines anderen Roboters erhalten kann.

Das Abschalten eines Roboters

Ein Spieler kann seinen Roboter abschalten und damit alle seine Schadenspunkte abgeben. Dies muss er allerdings eine Runde vorher ankündigen, und das Programm der aktuellen Runde wird auf jeden Fall vollständig durchgeführt. Der Roboter wird dann zu Beginn der neuen Runde abgeschaltet, das heißt er ist dann handlungsunfähig, kann aber weiterhin von Fabrikelementen und Robotern beeinflusst werden, sich also auch wieder

Schaden zuziehen. Bevor die Karten der nächsten Runde verteilt werden, kann der Spieler entscheiden, ob er seinen Roboter abgeschaltet lässt. Dann darf er wieder sämtliche Schadenspunkte abgeben.

Das Roboterduplikat ist beschädigt

Kommt der Roboter als Duplikat erneut ins Rennen, so erhält er zwei Schadenspunkte, weil ein Duplikat ja nie so gut sein kann wie das Original. Um mit der Erweiterung das Grundspiel sinnvoll ergänzen zu können, sollten alle diese Varianten mit ins Spiel einbezogen werden.

2.1.7 Optionskarten

Optionskarten nehmen eine Sonderrolle im Spiel ein und wurden erst mit der Erweiterung „Crash and Burn“ eingeführt. Verschiedene Optionskarten kann man in verschiedenen Situationen des Spiels einsetzen. Der Spieler kann seinem Roboter mit Hilfe dieser Karten weitere Eigenschaften geben. Beim Abschluss jeder Runde kann er sich, wenn sein Roboter auf einem Reparaturfeld mit zwei Schraubenschlüsseln steht, entscheiden, ob er zwei Schadenspunkte abgeben möchte, oder ob er eine Optionskarte vom verdeckten Stapel nehmen will. Diese muss er, wenn er sie vom Stapel aufnimmt, laut vorlesen und offen vor sich hinlegen. Der Spieler darf höchstens drei Karten besitzen, zieht er eine vierte, so muss er sich entscheiden, welche er ablegen will. Wird ein Roboter zerstört, so muss der Spieler eine Optionskarte abgeben.

Optionskartenbeeinflussung beim Austeilen der Programmierkarten

Hier können zwei Karten Einfluss ausüben:
„Zusätzlicher Speicherplatz“: Zu Beginn jeder Runde bekommt der Spieler eine zusätzliche Programmierkarte.
„Ersatz-Programm“: Einmal pro Runde darf der Spieler alle Karten, die ihm ausgeteilt wurden, zurückgeben, bevor er seinen Roboter programmiert hat, und erhält die gleiche Anzahl Karten neu vom Stapel. Dafür bekommt der Spieler dann einen Schadenspunkt.

Optionskartenbeeinflussung beim Programmieren der Roboter

„Befehl in Reserve“: Nachdem ein Spieler seine fünf Befehle programmiert hat, kann er eine sechste Karte verdeckt auf dieser Optionskarte ablegen. Diese kann er mit einer programmierten Karte austauschen, wenn diese noch nicht aufgedeckt wurde.

„Schwungrad“: Nachdem der Spieler fünf seiner Karten zur Programmierung verwendet hat, darf er eine sechste zur Seite legen und dann in der nächsten Runde zusätzlich verwenden.

Optionskartenbeeinflussung während des Programmablaufs - Befehle ausführen

„Notschalter“: Anstatt eine Programmierkarte aufzudecken, kann man diese mit dieser Optionskarte durch die oberste Karte des Stapels ersetzen. Danach müssen jedoch alle anderen Karten auch durch Karten vom Stapel ersetzt werden. Nach dem Abschluss der Runde wird die Funktion deaktiviert.

„Schutzschirm“: Der Schutzschirm kann den Roboter des Besitzers dieser Optionskarte vor Schaden aus einer bestimmten Richtung bewahren. Dieser Schutzschirm ist in jedem Programmschritt aktiv, und man muss sich vorher entscheiden, aus welcher Richtung man geschützt werden möchte. Dies macht man dadurch deutlich, dass man einen umgedrehten Schadenspunkt auf den Rand der Karte oben, unten, rechts oder links legt, je nachdem welche Seite geschützt werden soll. Falls die Richtung nicht auf diese Weise gekennzeichnet sein sollte, ist der Schutzschirm nicht aktiv.

„Drehscheibe“: Die Drehscheibe erlaubt einem Spieler, den Laser seines Roboters in die Richtung zu benutzen, die er vor einem Programmschritt festgelegt hat. Das gilt auch für die Benutzung des Turbo-Blasters, des Hochdruckstrahls, der Fernsteuerung, des Zufallsgenerators und des Traktorstrahls. Auch hier muss wieder mit Hilfe eines umgedrehten Schadenspunkts rechts, links oder unten angezeigt werden in welche Richtung sich die Drehscheibe bewegen soll. Liegt kein Punkt auf der Optionskarte, so schießt der Roboter nach vorn.

Auswirkungen erst nach dem Aufdecken der Programmierkarten

„Bremsen“: Wenn der Spieler den Befehl „Vorwärtsbewegung: 1 Feld“ programmiert hat, dann kann er mit dieser Optionskarte entscheiden, ob er sich ein Feld vorwärts bewegt, oder ob er stehen bleibt.

„Vierter Gang“: Bei dem Befehl „Vorwärtsbewegung: 3 Felder“ kann der Spieler wählen, ob er drei oder vier Felder vorwärts gehen möchte.

„Rückwärtsbeschleunigung“: Mit dieser Karte kann der Spieler wählen, ob er seinen Roboter ein oder zwei Felder rückwärts zieht, wenn er den Befehl „Rückwärtsbewegung“ programmiert hat.

„Mechanischer Arm“: Ist ein Programmschritt beendet, so kann der Roboter, wenn er auf einem der vier angrenzenden Feldern eines Checkpoints steht, den mechanischen Arm benutzen, um den Checkpoint zu berühren und damit ein Teilziel oder das Endziel der Robo Rally zu erreichen. Dabei sind diagonale Felder keine angrenzenden Felder. Der Arm kann durch eine Wand, aber nicht durch einen Roboter blockiert werden. Reparaturen oder die Aufnahme der Optionskarten können durch den mechanischen Arm nicht bewirkt werden.

„Rammbock“: Der Rammbock ist immer aktiv und fügt, wenn der Roboter einen anderen schiebt, diesem einen Schaden zu. Das tut er auch dann, wenn sich der Roboter nicht schieben lässt, weil er beispielsweise vor einer Wand steht.

Optionskartenbeeinflussung während des Programmablaufs - Fabrikelemente treten in Aktion

Mit den folgenden Karten werden die Auswirkungen, die bestimmte Fabrikelemente oder die Roboterlaser haben, verändert.

Expressbänder, Förderbänder und Zahnräder:

Im Folgenden die Optionskarten, die Auswirkungen auf Expressbänder, Förderbänder und Zahnräder haben.

„Kreiselstabilisator“: Wann immer der Besitzer dieser Optionskarte möchte, wird sein Roboter auf Express-, Förderbändern und auf Zahnrädern nicht gedreht.

Veränderungen am Laser der Roboter:

Mit diesen Optionskarten kann man die Laser der Roboter verändern:

„Doppellaser“: Der Roboter kann mit dem Doppellaser einem anderen Roboter zwei Schäden zufügen. Kombinationsmöglichkeit besteht mit der Karte Zielkontrolle und/oder Hochleistungslaser.

„Zielkontrolle“: Der Spieler muss bestimmen, ob er dem Besitzer des Roboters, den er beschädigt, eine Programmierkarte blockiert, oder ob er eine seiner Optionskarten zerstört. Diese muss der Spieler dann sofort auf den Ablagestapel zurücklegen. Kombinationsmöglichkeit besteht mit der Karte Doppellaser und/oder Hochleistungslaser.

„Hochleistungslaser“: Mit Hilfe dieser Optionskarte kann der Roboter durch eine Wand und durch einen Roboter schießen, um sein endgültiges Ziel zu treffen. Der Roboter, durch den der Laser schießt, wird dabei auch beschädigt. Kombinationsmöglichkeit besteht mit der Karte Zielkontrolle und/oder Doppellaser.

„Hecklaser“: Mit diesem verfügt ein Roboter dann über zwei Laser, einem nach vorne und einem nach hinten gerichteten. Mit beiden kann er anderen Robotern Schaden zufügen. Die Laser schießen gleichzeitig.

Anstelle der Laser einzusetzen:

Die folgenden Karten dürfen nur anstatt des Lasers des jeweiligen Roboters verwendet werden. Diese Eigenschaften ersetzen die Laser.

„Turbo-Blaster“: Wenn ein Roboter diesen benutzt, dann wird dem getroffenen Roboter ein Schaden zugefügt, und er wird ein Feld in Schussrichtung weiter geschoben. Wer die Karte erhält, bekommt gleichzeitig fünf Schadenspunkte ausgehändigt, die er auf diese Optionskarte legt. Jedes Mal, wenn er einen Roboter beschädigt, händigt er seinem Besitzer einen Schadenspunkt aus. Die Karte muss nach fünfmaliger Benutzung zurückgegeben werden, wenn alle Punkte abgegeben sind.

„Hochdruckstrahl“: Dieser schiebt einen anderen Roboter ein Feld in Richtung des Strahls, richtet aber keinen Schaden an.

„Fernsteuerung“: Mit dieser Karte wird es einem Spieler möglich, das Programm, mit dem sein eigener Roboter programmiert ist, auch auf einen anderen zu übertragen. Steht ein Roboter maximal sechs Felder in gerader Richtung vor dem eigenen Roboter, so kann er diesen fernsteuern. Wände und Roboter behindern die Verbindung nicht. Ab diesem Zeitpunkt bis zum Ende der Runde führen die beiden Roboter die identischen Befehle aus, wobei der steuernde Roboter seine Bewegungen immer als Erster durchführt.

„Zufallsgenerator“: Der nächste Befehl des Roboters, der von einem Laserstrahl getroffen worden wäre, wird mit Hilfe dieser Karte durch den Befehl der obersten Karte auf dem Stapel ausgetauscht. Im fünften Programmschritt ist der Zufallsgenerator nicht einsetzbar.

„Traktorstrahl“: Mit dem Traktorstrahl kann ein Spieler einen Roboter in die Richtung seines Roboters ziehen. Der Roboter wird hierbei nicht beschädigt. Steht dieser Roboter jedoch auf einem der vier benachbarten Felder, so darf die Optionskarte nicht benutzt werden.

Schaden:

Wenn ein Spieler für seinen Roboter einen Schadenspunkt erhalten soll, so kann er stattdessen einfach eine Optionskarte, die sich in seinem Besitz befindet, wieder zurückgeben. Mit den folgenden beiden Optionskarten kann der Schaden eines Roboters verhindert werden.

„Schutzhaut“: Diese kann verhindern, dass der Roboter Schadenspunkte erhält. Wird der Roboter des Besitzers dieser Karte beschädigt, so werden die Schadenspunkte, die er erhält, auf diese Optionskarte gelegt. Liegen auf dieser drei Schadenspunkte, so muss die Karte abgegeben werden.

„Schutzschilde“: Wenn ein Spieler seinen Roboter abgeschaltet hat, so ist er auf allen vier Seiten von je einem Schutzschild gesichert. Wird der Roboter wieder eingeschaltet, so verschwinden die Schutzschilde wieder.

Optionskartenbeeinflussung beim Abschluss der Runde:

Beim Einsetzen eines Roboterduplikats:
Setzt ein Spieler ein Duplikat seines Roboters ein, erhält er sofort zwei Schadenspunkte, es sein denn er besitzt die Optionskarte: **„Duplikat de Luxe“:** Der Spieler bekommt eine De-luxe-Ausführung seines Roboters, so dass er keine Schadenspunkte erhält.

Beim Abschalten eines Roboters:
„Stromkreisunterbrecher“: Besitzt ein Spieler diese Optionskarte, so muss er, wenn sein Roboter am Ende einer Runde drei oder mehr Schadenspunkte hat, seinen Roboter sofort für die folgende Runde abschalten. Die Ankündigung der Abschaltung entfällt hierbei.

2.1.8 Robo Rally 2005

In diesem Abschnitt sollen die Neuerungen im Vergleich zu den vorangegangenen deutschen Spieleversionen kurz erläutert werden.
Im Jahr 2005 erschien eine Neuauflage des Spiels Robo Rally bei Wizards of the Coast [Wiz05] erschienen. Allerdings ist diese Auflage nur in englischer Sprache erhältlich. Das Konzept ist jedoch dasselbe geblieben. Die Einzelheiten, die sich geändert haben, dienten uns zum Teil als Gedankenanstoß für die Realisierung des Spiels R_3D_3. Hier kurz die Neuerungen:

Checkpoints: Checkpoints sind nun durch Fähnchen gekennzeichnet. Ein eindeutiger Vorteil, denn man kann diese nun besser erkennen.

Roboter: Roboter haben jetzt alle Namen: Hammer Bot, Hulx X90, Spin Bot, Squash Bot, Trundle Bot, Twitch, Twonky, Zoom Bot (vgl. Abb. 2.4). Für jeden Roboter gibt es ein eigenes Program Sheet.

Abbildung 2.4: Die Roboter der Version 2005

Program Sheet: Hier können nun alle Programmierkarten, Lebens- und Schadenspunkte abgelegt werden. Wie oben erwähnt, erhält jeder Spieler für seinen eigenen Roboter einen eigenen Bogen, auf dem der Name des Roboters vermerkt ist.

Docking Bay Board: Dies ist ein zusätzlicher zweiseitiger Startspielplan, der es den Robotern ermöglicht auf gesonderten Feldern zu starten (vgl. Abb. 2.5 und 2.6). Jeder Spieler stellt seinen Roboter in ein Dock. Der erste Startplatz wird zufällig vergeben, alle anderen Docks werden dann im Uhrzeigersinn zugeteilt. Die Reihenfolge ist später wichtig, zum Beispiel wenn es darum geht, ob die Spieler ihren Roboter ausschalten

möchten. Wenn die Entscheidung eines Spielers seinen Roboter in der nächsten Runde abzuschalten davon abhängt, ob andere Spieler dies tun, kann hierzu jeder Spieler in der Reihenfolge der Docking-Startnummer gefragt werden, ob er seinen Roboter in der nächsten Runde abschalten möchte. Das Docking Board bietet den Vorteil, dass alle Roboter auf einem eigenen Feld starten können. Somit kann das Chaos zu Beginn, das in der deutschen Spieleversion immer entsteht, ein wenig entschärft werden, weil nicht alle Roboter auf demselben Feld starten müssen.

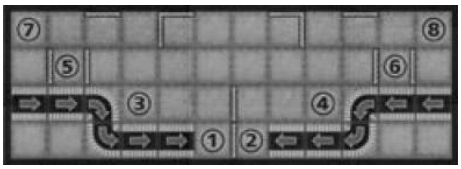


Abbildung 2.5: Docking Bay Board Seite 1

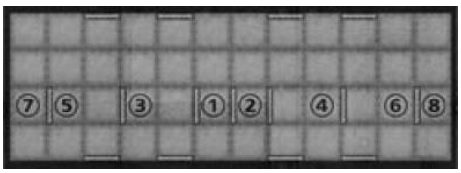


Abbildung 2.6: Docking Bay Board Seite 2

Timer: Wenn alle Spieler bis auf einen ihr Programm für die nächste Runde zusammengestellt haben, dann wird der Timer benutzt, der dann die Zeit angibt, die dem letzten Spieler noch verbleibt, um sein Programm fertig zu stellen. Gelingt ihm das nicht vor Ablauf der Zeit, so werden die restlichen Karten, im Extremfall sogar alle, verdeckt aus den Karten, die der Spieler erhalten hat, ausgewählt und die Register mit ihnen gefüllt.

Power Down Token: Jeder Spieler, der ankündigt seinen Roboter in der nächsten Runde ausschalten zu wollen, muss den Power Down Token auf sein Program Sheet legen.

Spielpläne und -varianten: Wizards of the Coast haben sich noch ein paar neue Spielmodi ausgedacht. So wird bei einer Version der Roboter „SuperBot“ gejagt, und von einem Spieler, der ihn endgültig zerstört hat, in Besitz genommen. Da er der einzige ist, der eine Checkpointfahne berühren darf, versuchen nun alle anderen, „SuperBot“ zu jagen und in ihren Besitz zu nehmen. Außerdem gibt es noch Teamvarianten, in denen Gruppen gegeneinander antreten können.

2.2 Unsere Realisierung des Spiels

In diesem Abschnitt werden die Teile der Spielregeln beschrieben, die nicht in die Realisierung aufgenommen werden.

2.2.1 Verwendete Varianten

In Abschnitt 2.1.6 wurde erwähnt, dass es mehrere Varianten gibt, mit denen man „Robo Rally“ spielen kann. Von diesen wird die Variante „Roboter besitzen Laser“ in die Realisierung übernommen.

2.2.2 Optionskarten

Bei einigen Optionskarten werden, da der Spielfluss nicht ständig unterbrochen werden soll, Entscheidungen über bestimmte Optionskarten bereits im Vorhinein der Runde von den Spielern festgelegt werden müssen. Dies wird bei der Optionskarte „Bremsen“, „Vierter Gang“ und „Rückwärtsbeschleunigung“ der Fall sein. Der Spieler legt vor Beginn der Runde fest, bei welcher Karte er gegebenenfalls bremsen, vier statt drei Schritte vorwärts gehen oder zwei statt einem Schritt rückwärts gehen möchte.

Um häufige Unterbrechungen des Spiels zu vermeiden, werden die Karten „Befehl in Reserve“, „Zielkontrolle“, „Notschalter“, „Schutzschirm“, „Drehscheibe“ von uns nicht übernommen. Die Spieler dürften bei den letzten drei Karten laut der „Robo Rally“ Regeln nach jedem Schritt entscheiden, ob sie diese im nächsten Schritt verwenden möchten, oder in welche Richtung sie die Karte verwenden möchten.

Optional werden die Karten „Turbo-Blaster“, „Krabbenbein“, „Dualer Prozessor“ eingebunden. Beim Turbo-Blaster wird dann die Benutzung auf 3 Runden beschränkt, da man diesen jeweils nur für eine ganze Runde einplanen kann.

2.2.3 Verwendete Neuheiten aus Robo Rally 2005

Die Version 2005 sieht ein Docking Bay Board als Startplatz für die Roboter vor. Diese Art Parkplatz wird auch in der Realisierung vorgesehen. Weitere Einzelheiten sind in Abschnitt 3.4.1.2 zu finden.

Des Weiteren wird die Idee der Fähnchen als Checkpoints übernommen. Diese sind auf einem solch großen Spielfeld, wie es konstruiert wird, viel besser zu erkennen. Detailliertere Ausführungen sind in Abschnitt 3.4.4.3 zu finden.

2.3 Das R_3D_3 Design

Für einen korrekten Spielablauf ist eine zentrale Instanz nötig, die als eine Art Schiedsrichter fungiert. Zu Beginn des Spiels wird auf dynamische Weise ein Host bestimmt, der Master wird. Dieser stößt die Befehlsverarbeitung an und bestimmt, wer welchen Befehl ausführen darf. Hierbei sind die Hauptaufgaben die Synchronisation und Konsistenzhaltung. Letzteres bedeutet, dass beispielsweise sichergestellt werden muss, dass alle Befehlskarten und Optionskarten im Spiel nur einmal existieren dürfen. Der Master

teilt in jeder Runde die Karten aus und muss außerdem dafür sorgen, dass die Fabrikelemente parallel abgearbeitet werden. Die restlichen Host sind Clients und werden vom Master angestoßen die Befehle auszuführen. Das jeweilige Spielfeld, auf dem der Roboter des Hosts steht, sendet die Befehle an den Roboter weiter. Auf die selbe Weise teilt ein Spielfeld dem Host dann zum Beispiel neue Schäden mit.

Die Informationen, um welches Fabrikelement es sich jeweils auf dem Feld handelt, auf dem der Roboter steht, werden im Spielfeld gespeichert, da es zuviel Datenaufwand erfordern würde, wenn der Roboter die Daten erst beim Spielfeld anfordern müsste. Des weiteren spart man Kommunikationsaufwand und Energie im Roboter ein, so dass eine höhere Akkulaufzeit erzielt werden kann.

Der in Kapitel 2.1 und 2.2 erwähnte Parkplatz erhält zu Beginn des Spiels eine weitere Aufgabe. Da es eine Anforderung des Spieles ist, dass die Spielfeldblöcke frei zusammensteckbar sein sollen, ist es nötig eine Initialisierungsphase zu starten, in der die Koordinatenstruktur für das Spiel festgelegt wird (siehe Kapitel 4.3.2.4). Diese Aufgabe übernimmt der im vorherigen Abschnitt erwähnte Parkplatz. Die Phase ist notwendig, damit im Verlaufe des Spiels zwischen den Hosts und den Feldern Nachrichten gesendet werden können. Am Ende dieser Phase kennt dann jeder Host die Adressen der Felder und jedes Feld die Adressen der im Spiel befindlichen Hosts. Im weiteren Spielbetrieb dient der Parkplatz jediglich als Stelle, an der die Roboter starten, und zu der alle toten Roboter während des Spiels geleitet werden, damit sie von dort aus eventuell wieder ins Spiel gelangen können. Ausführlichere Informationen zu der Struktur sind in Kapitel 4 zu finden.

3 Hardware

In diesem Kapitel wird die benötigte Hardware beschrieben. Dies ist im Einzelnen die Hardware des Roboters, des Spielfeldes und der Schnittstelle zum Host PC. Dabei haben diese Komponenten verschiedene Gemeinsamkeiten wie z.B. der Vorgang des Ätzens von Platinen.

3.1 Gemeinsamkeiten

Die einzelnen Hardwarekomponenten des Roboters, des Spielfeldes und der Host-PC Schnittstelle haben diverse Gemeinsame Teile. Diese sind wie folgt

- Erstellen von Platinenlayouts
- Ätzen von Platinen
- Mikrocontroller

Diese Teil werden hier im folgenden beschrieben.

3.1.1 Erstellen von Layouts

Zum Erstellen von Layouts für die eigenen Platinen verwenden wir das Programm EAGLE von Cadsoft. Hauptgrund dafür war die Verfügbarkeit einer Vollversion am Lehrstuhl. Außerdem erfüllt es alle gestellten Anforderungen für das Design unserer Platinen. Einige Bibliotheken mussten allerdings neu erstellt werden.

Einige Konventionen müssen allerdings eingehalten werden, damit beim Ätzen vernünftige Resultate erzielt werden können:

- **Leiterbahnbreite:** Die Leiterbahnen sollen mindestens 0,254mm (10mil) breit sein. Wenn möglich, sollten breitere Leiterbahnen benutzt werden.
- **Abstand von Leiterbahnen:** Der Abstand zwischen zwei Leiterbahnen oder zwischen einer Leiterbahn und einem anderen Signal soll ebenfalls nicht kleiner als 0,254mm (10mil) sein.
- **Lochgröße:** Löcher (auch Via und Pads) sollen mindestens einen Durchmesser von 0,8mm (32mil) haben.

Abweichungen sind z.B. bei den Pads für den Mikrokontroller oder den CPLD möglich.

3.1.2 Ätzen von Platinen

3.1.2.1 Platinenmaterial

Das Basismaterial einer Platine ist meist eine 0,5mm bis 1,5mm dicke, nicht leitende Schicht aus Epoxyd-Glashartgewebe oder Hartpapier. Als Materialien für die Platinen kommen auch noch andere genormte Stoffe in Frage. Eine detaillierte Auflistung dieser Stoffe findet sich z.B. unter www.pcb-pool.com. Darüber befindet sich eine 35 μm oder 70 μm dicke Kupferschicht. Auf die Kupferschicht ist Fotopositivlack aufgetragen, der dann mit einer lichtundurchlässigen Schutzfolie abgeklebt wird. Bei zweiseitigen Platinen sind die Kupfer-, Fotolack- und Schutzschicht auf beiden Seiten.

3.1.2.2 Herstellung von Platinen

Platinen lassen sich in 5 Schritten relativ einfach herstellen. Es gibt dabei mehrere Methoden, die einzelnen Schritte durchzuführen. Im Folgenden wird die zur Durchführung genutzte Methode beschrieben. Des Weiteren wird auch auf die vorhandenen Geräte und Mittel eingegangen.

Belichtung

Das mit Eagle erstellte Layout muss zunächst i.d.R. (die Oberseite spiegelverkehrt) auf eine transparente Folie gedruckt werden. Als Drucker kommt ein HP Deskjet 850C zum Einsatz. Dieser liefert brauchbare Ergebnisse. Für die Produktion von mehreren Platinen wird aber die Folie professionell durch eine Druckerei hergestellt.

Ist die Folie erstellt, wird die Platine ohne Abziehen der Schutzfolie auf die richtige Größe geschnitten. Daraufhin kann die Schutzfolie abgezogen und das Layout mit der bedruckten Seite nach unten auf die Platine gelegt werden. Bei zweiseitigen Platinen werden die beiden Folien zuerst übereinander gelegt und mit Tesafilm an drei Seiten fixiert. Die Platine kann dann hinterher einfach dazwischen geschoben werden.

Das Layout und die Platine werden danach auf die Glasplatte des am Lehrstuhl vorhandenen Belichtungsgerätes gelegt. Es ist darauf zu achten, dass die Glasplatte sauber ist und dass das Layout nicht verrutscht. Ist das Layout ordentlich platziert, kann die obere Folie geschlossen werden. Das Einschalten der Vakuumpumpe verhindert endgültig ein weiteres Verrutschen.

Nach einer weiteren Lagekontrolle der Folie kann der Deckel geschlossen und die Platine belichtet werden. Die Belichtungszeit hängt von der verwendeten Platine sowie vom Belichtungsgerät ab. Für das zur Verfügung stehende Gerät und die eingesetzten Materialen (Epoxyd-Fotoplatinen der Firma Bungard) ergibt sich eine optimale Belichtungsdauer von 2 Minuten und 30 Sekunden.

Entwickelung

Nach dem Belichten folgt das Entwickeln der Platine. Dabei werden die belichteten Lackteile abgelöst. Als Entwickler verwenden wir Natriumhydroxid-Lösung (Mischungsverhältnis 1:100). Die Platine kann (bei neuer Lösung) nach ungefähr 10-20 Sekunden aus der Lösung genommen und mit Wasser abgespült werden. Sollte die Lösung schon etwas älter und gut gesättigt sein, kann dieser Vorgang auch wesentlich länger dauern. Es sollte nun das komplette Layout sichtbar sein. Sind noch Lackreste vorhanden, so kann der Vorgang wiederholt werden. Fertig entwickelte Platinen sind an dem Glänzen der Kupferstellen, an denen der Fotopositivlack abgelöst wurde, zu erkennen. Diese Stellen sollten beim ersten Eintauchen in das Ätzbad auch sofort matt werden und den beginnenden Ätzprozess damit anzeigen. Ist dies nicht der Fall, so muss das Entwickeln wiederholt werden.

Nach dem Entwickeln sollte man die Platine auf Fehler kontrollieren. Unterbrochene Verbindungen können z.B. mit einem Edding oder Abreibesymbolen repariert werden.

Ätzen

Nach dem Entwickeln der Platine, folgt das Wegätzen der freigelegten Kupferschicht. Als Ätzmittel setzen wir Natriumpersulfat ein. Das Ätzmittel muss vor Beginn auf eine Temperatur von mindestens 40°C und höchstens 60°C gebracht werden. Das am Lehrstuhl vorhandene Ätzgerät besitzt einen Heizstab, der das Aufheizen übernimmt. Des Weiteren wird in den Ätzbehälter, in den die Platte senkrecht eingehängt wird, von unten Luft geblasen. Dies hat eine Beschleunigung des Ätzvorganges zur Folge.
Da die Ätzlösung durchsichtig ist, kann der Ätzvorgang relativ gut beobachten werden. Der Ätzvorgang dauert ca. 3-5 Minuten bei neuer Lösung. Ist die Platine fertig geätzt, so muss sie mit Wasser abgespült werden.

Säubern und Versiegeln

Nach dem Ätzen ist noch immer der Fotopositivlack auf der Platine vorhanden. Dieser kann leicht mit Spiritus entfernt werden. Zum Abschluss kann nun die Platine noch mit Lötlackspray behandeln werden. Dies verhindert die Oxidation der Leiterbahnen.

Bohren

Der letzte Schritt ist das Bohren. Wichtig dabei ist eine Bohrmaschine mit hoher Bohrerdrehzahl, die vorzugsweise in einem Bohrständer montiert ist. Dies ermöglicht ein verkantungsfreies Bohren mit einem dauerhaften, geringen Druck auf die Platine. Die gängigsten

Bohrer haben Größen zwischen 0,8 mm und 1,5 mm. Beim Einsatz von Glasfaserplatinen sollte ein Hartmetallbohrer verwendet werden, da normale HSS Bohrer schon nach wenigen Löchern stumpf werden.

3.1.3 Mikrocontroller

Die zentralen Steuerungsaufgaben in Spielfeld und Roboter übernehmen Mikrocontroller. In diesem Abschnitt werden die Auswahl und die Handhabung der eingesetzten Controller beschrieben.

3.1.3.1 Auswahl der Mikrocontroller

Anforderungen

Der konkreten Anwendungsfall der Projektgruppe hat mehrere Anforderungsbereiche: Roboter, Spielfeld, und die eventuell später geplanten Konsolen. Dies wirft die Frage auf, ob der gleiche Mikrocontroller in allen drei Bereichen zum Einsatz kommt oder ob für jeden Einsatzzweck ein spezialisierter Controller herangezogen werden soll.

Spielfeld Das Spielfeld, als zentrale Komponente des Spiels, übernimmt sicherlich den Großteil der Steuerungsaufgaben.

So muss unter Anderem der Spielplanaufbau initialisiert, die Befehle geroutet, die Roboterbewegungen koordiniert und die PC- bzw. Konsolenansteuerung realisiert werden. Entsprechende Programme müssen also hinterlegt und ausgeführt werden, wofür entsprechender Ressourcenbedarf zu berücksichtigen ist.

Weiterhin muss die Feld-zu-Feld-Kommunikation umgesetzt werden, wofür entsprechende Kommunikationsperipherie nötig ist. Auch um Befehle an die Roboter zu senden werden Kommunikationskanäle nötig sein. Hier sind LEDs mit entsprechenden Foto-Transistoren im Roboter vorgesehen, bei 5 Steuer-LEDs pro Feld und 9 Feldern pro Block ergäbe sich ein Ansteuerungsbedarf von 45 LEDs (vgl. Abschn. 3.4.6).

Die Grenzen bezüglich Abmessung und Stromverbrauchs sind sehr weit gefasst, da bei dem stationären Feld 15 cm x 15 cm zur Verfügung stehen (vgl. Abschn. 1.2.1.2) und auf Netzstrom zurückgegriffen werden kann.

Zur einzigen Einschränkung könnte die Belüftung werden, der Controller darf also keine zu große Leistungsaufnahme haben, da die aufgenommene Energie komplett in Form von Wärme wieder abgegeben werden muss.

Unter den Spielfeldblöcken nimmt der Parkplatz (vgl. Abschn. 3.4.1.2) eine besondere Rolle ein. Er ist als zentrale Steuereinheit für die Initialisierung des Spielfeldes zuständig

und hat weitere zusätzliche Aufgaben, die über die Funktionalität eines normalen Spielfeldblocks hinausgehen (vgl. Abschn. 4.2.1). Um allerdings Homogenität in Design von Hard- und Software bei allen Spielfeldblöcken beizubehalten, wird der Parkplatz mit dem gleichen Controller ausgestattet werden, wie auch die übrigen Spielfeldblöcke, daher wird der Parkplatz im Folgenden nicht weiter einzeln betrachtet.

Roboter Bei den Robotern stehen sicherlich Abmessung und Stromverbrauch an oberster Stelle. Es werden ca. 5 cm x 4 cm Platz für die Platinen zur Verfügung stehen (vgl. Abschn. 1.2.1.1 und 3.3.5), wobei noch Motoren, Mechanik und Stromversorgung berücksichtigt werden müssen.

Der Roboter braucht aber auch weniger Rechenleistung, da seine generelle Aufgabe wahrscheinlich im einfachen Entgegennehmen und Ausführen von Befehlen bestehen wird (vgl. Abschn. 1.2.1.1 und 3.3.1). Hier müssen die oben erwähnten Foto-Transistoren abgefragt und die entsprechenden Befehle an die Mechanik und Motoren weitergegeben werden, wobei Digital/Analog-Wandler vorteilhaft sein könnte.

Unter dem Aspekt der Energieversorgung eignet sich eine 3 V-Lösung, da wahrscheinlich auf Batterien oder Akkus zurückgegriffen wird. Die maximale Stromaufnahme sollte ca. 150 mA nicht übersteigen (vgl. Abschn. 3.3.2.4). Weiterhin sind Stromsparmodi essentiell, da der Roboter die meiste Zeit im Wartezustand verbringen wird.

Konsolen Die Konsolen sollen auf graphischer Basis bedient werden, deswegen wäre eine dorthingehende Unterstützung zur Erleichterung der Umsetzung wünschenswert, zum Beispiel in Form einer integrierten LCD-Ansteuerung.

Je nach Ansteuerungsart der Konsole muss dementsprechend ein Kompromiss zwischen Rechenleistung und Energiebedarf gefunden werden. Wird die Konsole per Kabel mit dem Spielfeld verbunden, kann so auch die Stromversorgung gesichert werden. Es muss also keine gesonderte Rücksicht auf den Energieverbrauch genommen werden, da in diesem Fall auf Netzstrom zurückgegriffen werden kann. Anders ist dies bei einer drahtlosen Ansteuerung: Es entfällt nicht nur die netzgebundene Stromversorgung, sondern es kommen noch die Energiekosten für ein Drahtlos-Modul hinzu, welches dann auch idealerweise in dem passenden Controller integriert sein sollte.

Mikrocontroller-Varianten

Bei der Vielzahl der verfügbaren Mikrocontroller existieren mehrere Prozessorvarianten, die je nach Einsatzzweck verschiedene Merkmale haben. Im Folgenden werden die wichtigsten beleuchtet und im Hinblick auf unseren Einsatzzweck bewertet. Im Falle einer inhomogene Lösung (verschiedene Mikrocontroller für die verschiedenen Anforderungsbereiche) wäre es allerdings sinnvoll, die drei Controller aus derselben Typfamilie zu nehmen, um diese mit denselben Entwicklungswerkzeugen programmieren zu können.

Embedded Processors Der ARM-Prozessor-Kern [ARM05] ist ein eingebetteter, frei programmierbarer Mikroprozessor, der mit 8, 16 und 32 Bit Datenbusbreite erhältlich ist. Er beherrscht Pipelining und eine Form der Code-Komprimierung (thumb). Es existieren viele Allzweck-Lösungen, so dass ein ARM für eine homogene Lösung ideal ist. Da ARM der Markführer im Feld der eingebetten Prozessoren ist, existieren vielerlei Entwicklungs- und Programmierwerkzeuge (toolchains) für diesen Kern, unter anderem ist eine komplette Toolchain am Lehrstuhl vorhanden. Externe Programmierumgebungen existieren z.B. von Keil [Kei05].

Der AVR RISC [Atm05g] ist eingebetteter Prozessor mit 8 Bit Datenbusbreite. Er hat aber dafür einen von Haus aus geringen Stromverbrauch. Viele nützliche Ausstattungsmerkmale, wie Stromsparmodi und In-System-Programming und -Debugging sind standardmäßig enthalten. Atmel bietet diesen Prozessor in vielen vordefinierten, spezialisierten Umgebungen an, so dass dieser Typ eher für einen inhomogenen Ansatz in Frage kommt. Die voll integrierte Entwicklungsumgebung liefert Atmel in Form des AVR Studios, wobei hier alle Tools in eine Plattform integriert sind.

DSP Ein Digitaler Signal Prozessor ist auf Gleitkomma-Verarbeitung ausgelegt und besitzt deshalb häufig zusätzliche Multiplizier-ALUs und Parallelverarbeitungsansätze wie VLIW-Befehlssätze (very large instruction words). Aufgrund dessen werden DSPs häufig in Multimedia-Anwendungen eingesetzt, wo schnelle, häufig sogar in Echtzeit stattfindende, Gleitkomma-Berechnungen gefordert sind, zum Beispiel bei Musik- oder Video-Dekodierung. Auch gibt es kombinierte Ansätze aus eingebettetem Prozessor mit unterstützender DSP-Einheit. Da die Anforderungen Anwendungen von Multimedia in Form von Bild (Videos oder Animationen) oder Ton (Musik- oder Sound-Wiedergabe) nicht vorsehen, wäre ein DSP ein unpassender Ansatz.

FPGA Field Programmable Gate Arrays bieten den Vorteil, dass sie dynamisch umprogammierbar sind, und dann die gestellten Aufgaben in Hardware lösen. Da allerdings diese Flexibilität nicht unbedingt notwendig ist und die Verwendung von FPGAs auch Nachteile bei der Integration einbringt, ist dieser Ansatz auch zu verwerfen.

Peripherie

Nicht nur der zentrale Prozessor bestimmt das Einsatzgebiet eines Mikrocontrollers, sondern maßgeblich auch seine Peripherie, da sie darüber entscheidet, welche Funktionalität bereits in Hardware vorhanden ist, und somit nicht mehr in Software entwickelt werden muss.

Es gibt eine Vielzahl von Erweiterungen, und wie so oft gibt es in mehreren Bereichen unterschiedliche Ansätze für dieselbe Funktionalität. Andererseits ist es gerade bei Kommunikationsprotokollen üblich, einen zentralen seriellen Baustein zu verwenden, der die

Kommunikation steuert, was zur Folge hat, dass die Standards (je nach Hersteller) zueinander kompatibel sind, oder einfach ineinander übersetzt werden können.

Nachfolgend werden die für uns wichtigsten Peripherie-Elemente kurz erläutert:

USART Universal Synchronous/Asynchronous Reciever Transmitter ist eine serielle bidirektionale Schnittstelle zum Datenaustausch. Sie erlaubt Funktionen wie Handshake, Paritätsüberprüfungen, Flusskontrolle und ähnliches.

TWI Two Wire Interface ist eine Byte-basierte serielle Schnittstelle, die einen sehr einfachen Aufbau hat: Sie besteht aus zwei Verbindungen, und zwar einer Takt- und einer Daten-Leitung. So werden die Daten zwischen zwei Endstellen seriell und getaktet übertragen.

PWM Pulse Width Modulation ist eine zeitbasierte Informations Codierungsart: Innerhalb jedes Zeitfensters wird ein je nach Information entsprechend langer Impuls erzeugt, der entweder zeitgleich mit dem Zeitfenster beginnen kann, oder innerhalb desselben zentriert wird. Durch diese Art der Codierung kann auf einfache Weise ein Gleichstrom-Motor angesteuert werden, der sich umso schneller dreht, je länger die Impulse innerhalb der Zeitfenster sind.

A/D-Wandler Die Analog/Digital-Wandler auf den gängisten Mikrocontrollern haben meist einen 10 Bit breiten Ausgang, und je nach Ausstattungsvariante auch mehrere Kanäle. Dies bedeutet, dass mehrere eingehende Analog-Signale unabhängig voneinander umgewandelt werden können.

General Purpose I/O Lines Viele Controller besitzen auch noch eine gewisse Anzahl von general purpose lines, die im Prinzip direkt ansprechbare Ein- und Ausgangs-Pins des Controller-Bausteins sind. Meist sind sie 5 V-ansteuerbar, auch wenn der Chip an sich nur mit etwa 3 V betrieben wird, so dass die einkommenden Signale nicht erst vorbearbeitet werden müssen. Durch die Anzahl an GP I/O Lines lässt sich gut eine parallele Kommunikation realisieren, so können z. B. mehrere LEDs oder Segmentanzeigen gleichzeitig angesteuert werden.

Verschiedene mögliche Kandidaten

Im folgenden werden einige zu den oben genannten Anforderungen passende Mikrocontroller speziell im Hinblick auf die vorgestellten Merkmale aufgelistet. Dabei werden sowohl Controller die alle drei Aufgabenbereiche übernehmen können, als auch durch verschiedene Controller realisierte Lösungen vorgestellt.

Philips LPC2114 [Phi03]

- ARM, 60 MHz
- 16 kB SRAM
- 128/256 kB Flash
- 2 USART
- 1 TWI
- 1 PWM-Wandler
- 4-Kanal 10 Bit A/D-Wandler
- 46 GP I/O Lines
- Größe: 12 mm x 12 mm

Was den LPC2114 auszeichnet, ist seine vergleichsweise hohe Taktrate und viel SRAM-Speicher bei wenig Leistungsaufnahme. Sein Stromsparmodus durch gezieltes Abschalten einzelner Komponenten kann hilfreich sein. Auch sind genügend Kommunikationsschnittstellen vorhanden; PWM- und A/D-Wandler erleichtern die Motorsteuerung, zusätzlich lassen sich etwa LEDs über die GP I/O ansteuern. Die Größe ist auch für den Einsatz im Roboter annehmbar, also ein guter Kandidat für eine homogene Lösung.

Atmel AT91M42800A [Atm05a]

- ARM, 33 MHz
- 8 kB SRAM
- 2 USARTs
- 54 GP I/O Lines
- Größe: 22 mm x 22 mm

Nachteil dieser Atmel-Lösung wäre, dass der Code außerhalb des Controllers hinterlegt wird und über einen externen Bus darauf zugegriffen werden muss. Außerdem bieten sich aufgrund dessen nur sehr eingeschränkte Debugging-Maßnahmen. Ausreichende Kommunikationsperipherie ist vorhanden, gerade die hohe Zahl der GP I/O Pins und die damit einhergehende Größe sprechen dafür, ihn eher als separate Spielfeldlösung einzusetzen, zumal keine Motoransteuerungen vorhanden sind.

Atmel AT91M55800A [Atm05b]

- ARM, 33 MHz
- 8 kB SRAM

- 2 USARTs
- 8-Kanal 10 Bit A/D-Wandler
- 58 GP I/O Lines
- Größe: 26 mm x 26 mm

Auch bei diesem Controller besteht der Nachteil des externen Codes, der per Bus Controller aufgerufen wird. Kommunikationsperipherie ist vorhanden, auch in Form eines 2-Kanal 10 Bit D/A-Wandler. Insgesamt also eine minimalistische Lösung für das Spielfeld, aufgrund der Größe ungeeignet für den Roboter.

Atmel AT91RM3400 [Atm03]

- ARM, 66 MHz
- 96 kB SRAM
- 4 USARTs
- 1 TWI
- Größe: 16 mm x 16 mm

Ein schneller Mikrocontroller mit großem Speicher und viel Peripherie, so zum Beispiel einen USB 1.1 Controller und sogar ein MultiMediaCard-Interface. Der Code wird allerdings beim Booten extern geladen, da kein Flash-Speicher verfügbar ist. Dieser Controller würde sich für eine Luxus-Lösung anbieten, da er üppig ausgestattet ist.

Atmel AT91SAM7S32 [Atm05c]

- ARM, 55 MHz
- 8 kB SRAM
- 32 kB Flash
- 1 USART
- 4-Kanal 16 Bit PWM-Wandler
- 8-Kanal 10 Bit A/D-Wandler
- 21 GP I/O Lines
- Größe: 7 mm x 7 mm

Ein kleiner und sparsamer Mikrocontroller, der annehmbar ausgestattet ist. Wegen seiner eher eingeschränkten Kommunikations- aber vielfältigen Steuerungsperipherie als Roboter-Controller gut denkbar.

Atmel ATmega6490 [Atm05e]

- AVR, 8 MHz bzw. 16 MHz
- 4 kB SRAM
- 64 kB Flash
- 1 USART
- 8-Kanal 10 Bit A/D-Wandler
- 68 GP I/O Lines
- Größe: 16 mm x 16 mm

Ein AVR-Chip mit zusätzlicher LCD-Ansteuerung und weiteren Extras wie Universal Serial Interface und Analog-Comparator. Dies ist Beispiel für einen Controller mit Grafik-Komponente, wobei hier das High-End-Modell der Reihe gewählt wurde.

Atmel ATmega162 [Atm05d]

- AVR, 16 MHz
- 1 kB SRAM
- 16 kB Flash
- 2 USARTs
- 6-Kanal PWM-Wandler
- 35 GP I/O Lines
- Größe: 12 mm x 12 mm

Ein durchschnittlich ausgestatteter Chip, diesmal aus der AVR-Familie. Leicht unterdurchschnittliche Kommunikationsmöglichkeiten und Steuerungsperipherie (fehlender A/D-Wandler) werden auch durch einen Analog-Comparator nicht ausgeglichen. Eher eine Notlösung für Feld und Roboter.

Atmel AT90PWM2 [Atm06a]

- AVR, 16 MHz
- 512 Bytes SRAM
- 8 kB Flash
- 1 USART
- 1 PWM-Wandler

- 10 Bit A/D-Wandler
- Größe: 15 mm x 7 mm

Dieser Chip wurde speziell für Motoransteuerungen entwickelt und enthält deswegen eine zusätzliche ALU, die schnell multiplizieren kann, einen 10 Bit D/A-Wandler und 3 Analog-Comparators, dafür aber sehr eingeschränkte Kommunikationsmöglichkeiten. Dieser Controller käme für den Einsatz im Roboter in Frage.

Atmel ATtiny26L [Atm05f]

- AVR, 1 MHz bzw. 16 MHz
- 128 Bytes SRAM
- 2 kB Flash
- 1 PWM-Wandler
- 10 Bit A/D-Wandler
- 16 GP I/O Lines
- Größe: 25 mm x 8 mm

‚Tiny' heißt klein: Dementsprechend sind auch Ausstattung und Stromverbrauch. Als einzige Kommunikationsperipherie steht ein Universal Serial Interface zur Verfügung, dafür gibt es passende Motorsteuerungen, auch ein Analog Comparator ist vorhanden. Also auch ungeeignet für den Einsatz im Spielfeld, nur als Roboter-Controller einsetzbar.

Auswahl der Controller für Spielfeld und Roboter

Aufgrund der verschiedenen Anforderungen der Einsatzbereiche Spielfeld und Roboter im Hinblick auf Größe, Stromverbrauch, Kommunikations- und Steuerungsaufgaben ist es schwierig eine Kompromisslösung zu finden, die beiden Bereichen zufriedenstellend gerecht wird. Daher ist es sinnvoll die inhomogene Lösung zu wählen und für Spielfeld und Roboter unterschiedliche Controller zu nehmen.

Es bietet sich an, ARM-basierte Controller zu verwenden, da diese länger auf dem Markt sind und somit eine größere Auswahl an Entwicklungswerkzeugen und breiter gefächerte Produkterfahrungen zur Verfügung stehen. Darüber hinaus ist eine Toolchain am Lehrstuhl bereits vorhanden. Auch ist die Programmierung in der 16/32 Bit ARM-Umgebung wesentlich komfortabler, als auf den 8 Bit AVR Kernen. Zudem sind die AVR-Chips nur knapp mit Speicher ausgestattet, da sich der spätere Bedarf im Vorfeld nur schwer abschätzen lässt, sollte in diesem Punkt lieber großzügiger kalkuliert werden.

Die beste Lösung im Bezug auf Speicherausstattung, Kommunikationsperipherie und Rechenleistung ist zweifelsfrei der Philips LPC2114, wobei hier noch auf die größere Variante mit 256 kB Flash-Speicher zurückgegriffen werden kann.

Da die Hauptanforderungen des Roboters an den Controller Größe und passende Steuerungsperipherie sind, fällt das Augenmerk sofort auf dem AT91SAM7S32, als einziger denkbarer Controller der vorgestellten ARM-Chips. Da auch er eine zufriedenstellende Speicherausstattung ausweist, wird er im Roboter zum Einsatz kommen.

3.1.3.2 Compiler-Tools

Compiler wandeln Quellcode eines Programms in ausführbaren Code um. Dabei ist die Compile- sowie die Ziel-plattform i.d.R. diesselbe. Hier wird jedopch ein Cross-Compiler benötigt. Dieser kompiliert den Code auf einer Windows- oder Linux-Plattform. Der resultierende Code ist dabei aber nur auf dem jeweiligen Controller lauffähig. Diese sind beide Controller vom Typ `arm7tdmi`.

Zur Auswahl stehen hauptsächlich 2 verschiedene Compiler-Tools. Zum einen sind es die `ARM` Compiler-Tools und zum anderen die `GCC` Compiler-Tools. Dabei fällt schliesslich die Einigung auf die `GCC` Compiler-Tools, da diese kontinuierlich weiterentwickelt werden, kostenlos und zudem weit verbreitet in der OpenSource-Community sind.

Folgende Schritte müssen durchlaufen werden, um aus dem Quellcode die ausführbaren Daten zu erstellen:
`Compilieren` ≻ `Linken` ≻ `BIN/HEX extrahieren`

Mit Version 4 der GCC-Tools sind 2 Befehle verantwortlich:

```
gcc -combine -mcpu=arm7tdmi -O <linker_options> -o compiled.elf <objects>
```

```
objcopy -O [ihex | binary] compiled.elf output.[hex | bin]
```

Dabei gibt beim ersten Befehl *<objects>* die zu compilierenden Dateien an. Durch die Angabe von `-combine` kann man sogar alle Dateien des Projektes angeben, dabei versucht der Compiler, diese möglichst gut zu optimieren, falls Optimierungen gewünscht sind. Die *<linker_options>* geben Optionen für den Linker an. Unter anderem wird dort eine Datei `linker.ld` übergeben. Diese enthält Informationen über den Speicheraufbau und bestimmt, wo RAM und wo ROM ist. Alles zusammen wird im `.ELF` Format gespeichert. Der zweite Befehl extrahiert den ausfürbaren Teil aus der `.ELF` Datei und speichert entweder im reinen `binary` Format oder im `HEX` Format, je nach weiterer Verwendung.

3.1.3.3 Integrated Development Environments

Es gibt einige Entwicklungsumgebungen (IDE - Integrated Development Environment) für C/C++. Neben Microsoft Visual Studio, KDeveloper und einer ganzen Reihe von

mehr oder weniger bekannte IDEs gibt es auch für die ARM-Prozessoren spezialisierte IDEs. Dazu gehört auch das Programm `Keil uVision` [Kei]. Dies wird hier vorgestellt.

Von der Keil Homepage müssen die Installer für `Keil uVision` und für die `GNU-Tools` heruntergeladen und installiert werden. Dabei ist die Version von der Homepage zu nehmen und keine von GCC direkt.

Neues Projekt erstellen
Unter `Project > New Project` kann ein neues Projekt erstellt werden. Die Frage, ob der Startup-Code für den ausgewählten Controller ins Projektverzeichnis kopiert werden soll, kann mit 'Ja' beantwortet werden. (siehe Abb. 3.1)

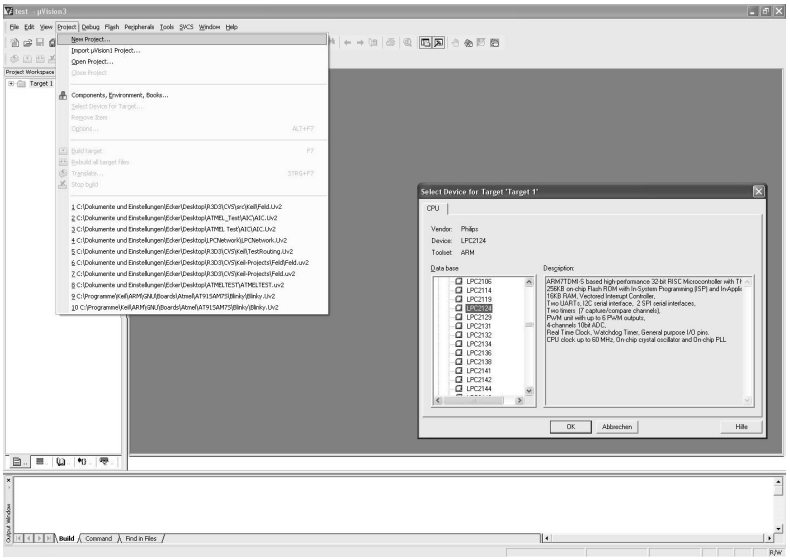


Abbildung 3.1: Keil uVision: Neues Projekt anlegen

Da sich der Startup-Code für die GNU-Tools unterscheidet, müssen wir aber diesen Startcode ersetzen. Das File `Startup.S` im Projektverzeichniss muss durch

`PROGRAMME/Keil/ARM/GNU/Boards/Keil/MCB2100/BlinkyCAN/Startup.S`

ersetzt werden. Die Dateien `ram.ld` und `flash.ld` könen direkt mitkopiert werden, sie werden für den Linker benötigt.

Einrichten
Beim erstmaligen Nutzen von `Keil uVision` müssen die GNU-Optionen eingestellt werden. Das entsprechende Fenster ist unter
`Project > Components, Environment and Books...` zu erreichen.
Hier müssen die GNU-Tools wie in Abb. 3.2 zu eingestellt werden.

In Abb. 3.3 und Abb. 3.4 sind die Optionen für den Linker und das Ausgabeformat zu sehen. Dies ist unter `Project > Options` zu erreichen.

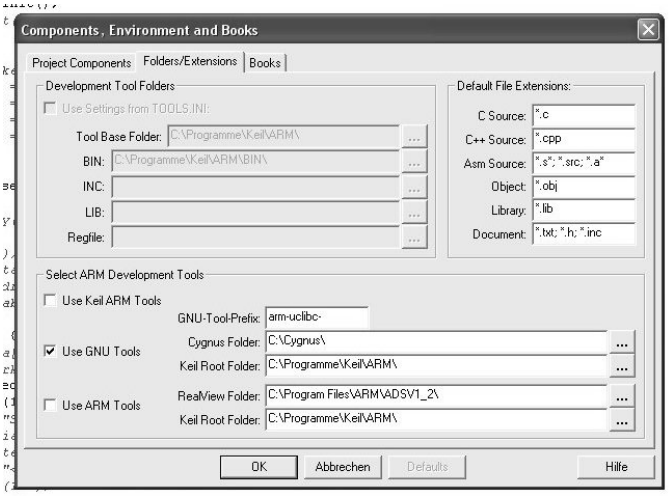


Abbildung 3.2: Keil uVision: GNU Tools einstellen

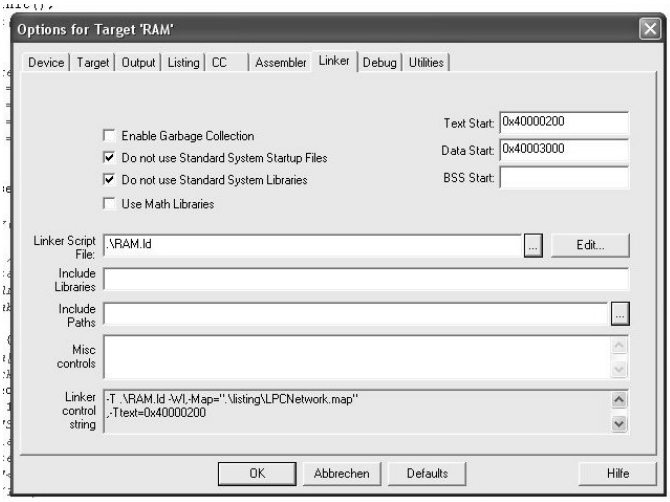


Abbildung 3.3: Keil uVision: Linker einstellen

3.1.3.4 Ausführung im Controller

Nach dem Kompilieren und Linken des Codes ist eine ausführbare Datei im `ELF` Format vorhanden. Das Flashtool von Philips für den LPC2114/2124 benötigt den Code im `hex` Format, hingegen das Flashtool von ATMEL für den AT91SAM7S32 benötigt den Code im `binary` Format. Das Tool `arm-elf-objcopy` kann den ausführbaren Code aus dem `ELF` Format extrahieren und wahlweise im `binary` oder `hex` Format speichern.

```
arm-elf-objcopy -O ihex input.elf output.hex
```

```
arm-elf-objcopy -O binary input.elf output.bin
```

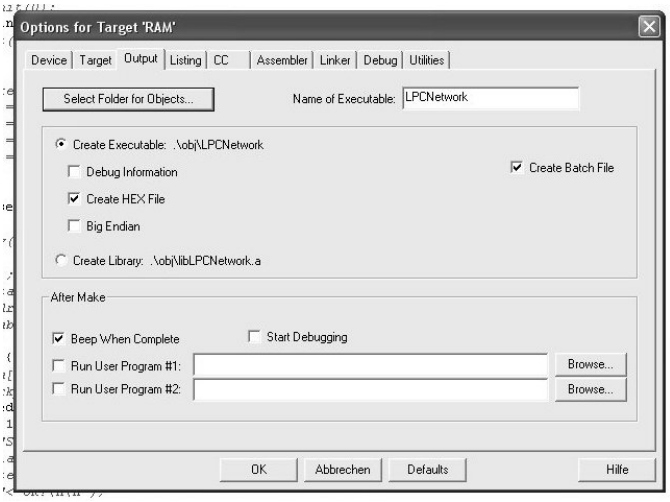


Abbildung 3.4: Keil uVision: Hex Ausgabe aktivieren

Flashen des Philips LPC2114/2124

Zum Flashen auf den LPC2114/2124 wird das `LPC2000 Flash Utility` von Philips benötigt. Dieses steht kostenlos zum Download beim Hersteller zur Verfügung. Nach der Installation und Aufruf des Programms sehen wir das Fenster wie in Abb. 3.5 zu sehen. Folgende Einstellungen sind wichtig anzupassen:

- Device
- XTAL Freq.
- Connected to Port
- Use Baud Rate (siehe [Phi03], Seite 226, Tabelle 166)

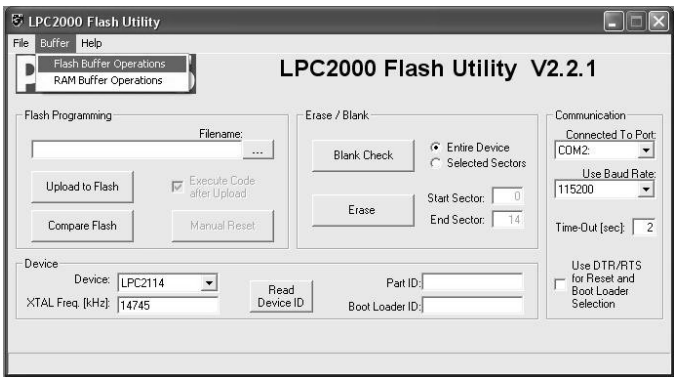


Abbildung 3.5: Hauptfenster des LPC2000 Flash Utility

In Abb. 3.6 ist das Flashfenster zum Upload der Daten in den Flashspeicher gezeigt. Zuerst muss der ausführbare Code im `hex` Format eingelesen werden. Ganz wichtig ist `Vector Calc` durchzuführen, da erst dann der Controller den Code als gültig erkennt. `Vector Calc` berechnet einen Wert aus den Interruptvektoren, die ganz am Anfang des Speichers angeordnet sind. Anschliessend kann der geladene Code durch Drücken der Taste `Upload to Flash` in den Flashspeicher kopiert werden.

Ein ähnliches Fenster steht auch für den Upload ins RAM zur Verfügung. Dabei muss aber kein `Vector Calc` durchgeführt werden, da der Code manuell aus dem Flash Programm aufgerufen wird. Dazu muss der Code entsprechend gelinkt werden wie in Abschnitt 3.1.3.2 beschrieben. Daraufhin kann bei Angabe der richtigen Startadresse der Code direkt aus dem RAM gestartet werden.

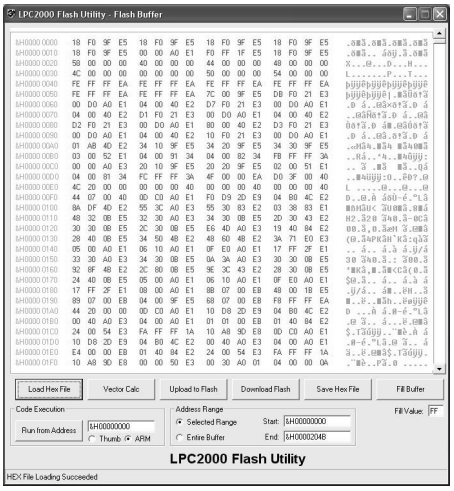


Abbildung 3.6: Uploadfenster für den Flashspeicher

Flashen des ATMEL AT91SAM7S32

Zur Programmierung des AT91SAM7S32 stellt Atmel ein Flash-Tool namens SAM-BA zur Verfügung [Atm06b], das den Controller sowohl über die serielle Schnittstelle, als auch über USB ansprechen kann (vgl. Abb. 3.7). Da auf dem Testboard die Debug-Unit (DBGU) des Controllers mit einer seriellen Schnittstelle verbunden ist (vgl. Abschn. 3.3.4), betreiben wir das SAM-BA Flash-Tool per DBGU-Connection über einen COM-Port, der mit der Debug-Unit (DBGU) des Controllers verbunden ist. Dabei ist zu beachten, dass das Tool nur über einen der ersten vier COM-Ports kommunizieren kann, wobei COM3 (sofern unbelegt) ein interner Loop-Back Port ist, über den man das Tool auch starten kann, ohne dass ein Controller angeschlossen ist.

Ansonsten liefert das Tool die Fehlermeldung „Memory File not found“ (vgl. Abb. 3.8), die angibt, dass die Verbindung zum Controller nicht hergestellt werden konnte.

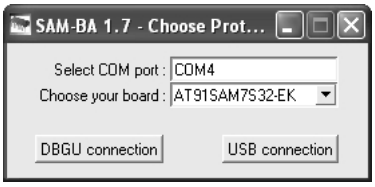


Abbildung 3.7: Auswahlbildschirm des SAM-BA Flash Tools

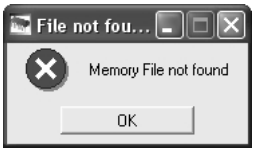


Abbildung 3.8: Fehlermeldung des SAM-BA Flash Tools

Voraussetzung für das Verwenden des SAM-BA Flash Tools ist die richtige Vorbereitung des Mikrocontrollers: Auf diesem muss das Programm SAM-BA Boot hinterlegt sein, das bei Ausführung nach einem Reset die serielle Schnittstelle controllerseitig initialisiert. Der SAM-BA Boot Code wird durch Setzen bestimmter Pins beim Starten des Controllers aus dem internen ROM in den Flash Speicher kopiert (SAM-BA Boot Recovery, vgl. [Atm05c]). Insbesondere beim ersten Booten des Controllers muss dieser Vorgang ausgeführt werden, da dieser sonst nicht mit dem SAM-BA Flash-Tool angesprochen werden kann. Weitere Ursache der Fehlermeldung in Abbildung 3.8 kann auch sein, dass der Controller zurückgesetzt werden muss, um wieder den internen SAM-BA Boot Code auszuführen.

Nach dem Starten und einer erfolgreichen Verbindung zum Mikrocontroller öffnet sich der Hauptbildschirm (vgl. Abb. 3.9).

Hier können folgende Aktionen ausgeführt werden:

- Auslesen des Speichers im oberen Übersichtsfenster, oder in eine Datei mittels „Recieve File“
- Hochladen von Code und Daten in den Speicher an eine bestimmte Adresse mittels „Send File“. Hierbei ist zu beachten, dass die Auswahl des Speichers, in den geschrieben werden soll, beim Starten des Flash-Tools stets auf „Flash“ voreingestellt ist. Dementsprechend kann die unerwartete Warnung „At least one lock region is locked!“ (vgl. Abb. 3.10) darauf hindeuten, dass versehentlich in den Flash-Speicher geschrieben werden soll. Damit besteht die Gefahr, dass die Speicherbereiche, die vom SAM-BA Boot Code benutzt werden, überschrieben werden, und so der Mikrocontroller nicht mehr über die serielle Schnittstelle angesprochen werden kann, bis die SAM-BA Boot Recovery Routine wieder ausgeführt wurde.
- Manuelle Befehlsausführung: Starten von Code ab einer bestimmten Adresse. Dazu wird der Befehl *go* verwendet.

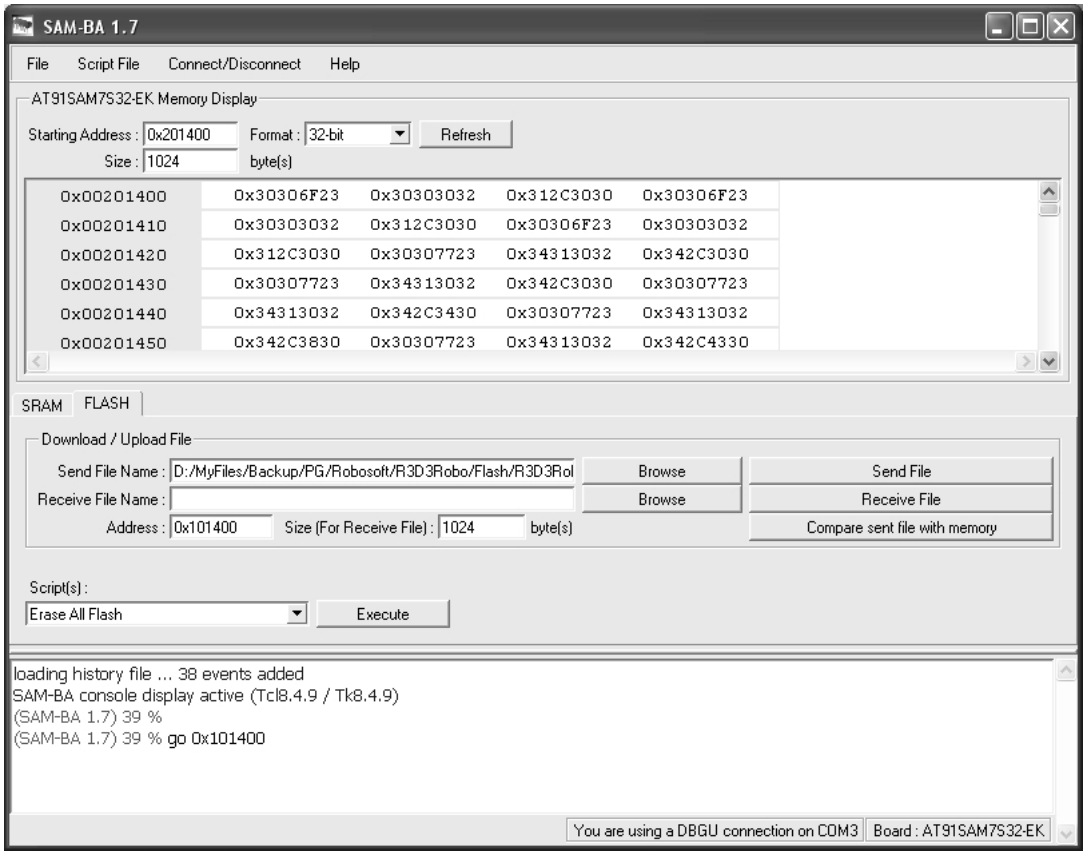


Abbildung 3.9: Hauptbildschirm des SAM-BA Flash Tools

3.2 Kommunikation

Der nötige Informationsaustausch unter den einzelnen Komponenten erfolgt auf unterschiedliche Arten und Weisen. Je nachdem welche Komponenten miteinander kommunizieren, sind unterschiedliche Anforderungen an die Hardware und Software gestellt. Es müssen 3 verschiedene Kommunikationsverbindungen vorhanden sein, sodass alle Komponenten untereinander Informationen austauschen können.

3.2.1 Host PC - Spielfeldbox

Die Kommunikation zwischen Host PC und Spielfeldbox sollte über eine bereits vorhandene Schnittstelle am PC erfolgen. Eine gemeinsame Schnittstelle zwischen Controller und PC ist die Serielle Schnittstelle (RS-232). Da die Spielfeldboxen eine eigene Schnittstelle verwenden (siehe Abschnitt 3.2.2), ist es nötig, einen Adapter zwischen PC und Spielfeld zu setzen. Dieser ist detailliert in Abschnitt 3.6 beschrieben. Seine Aufgabe ist es ankommende Pakete in beide Richtungen weiterzuleiten.

Abbildung 3.10: Lock-Warnung des SAM-BA Flash Tools

3.2.2 Spielfeldbox - Spielfeldbox

Informationen müssen von allen Spielfeldboxen versendet und empfangen werden können. Jede Spielfeldbox hat bis zu 4 Nachbarn, mit denen Informationen direkt ausgetauscht werden können. Dazu sind 4 Kommunikationskanäle notwendig. Die Verarbeitung und Weiterleitung der Informationen zum jeweiligen Ziel erfolgt im Protokoll-Stack, siehe dazu 4.3. Hier wird nur die Kommunikationshardware zwischen den Spielfeldboxen behandelt.

Zur Übertragung von einzelnen Bits auf einem Kanal werden jeweils 2 Leitungen verwendet, eine DATA- und eine CLOCK-Leitung. Die Daten sind digital codiert, hier kann eine Leitung nur den Zustand 0 (etwa 0 Volt) und Zustand 1 (etwa 3,3 Volt) einnehmen. Die Pull-up Widerstände sind mit der 3,3V Versorgungsspannung verbunden, sodass im Allgemeinen die Leitungen logisch 1 aufweisen. Diese sind mit R1-R4 in Abbildung 3.11 gekennzeichnet. Zur Kurzschlusssicherheit der Controller sind zusätzliche Widerstände R5-R8 in Abbildung 3.11 eingebaut. Zusätzlich sind als Überspannungsschutz Dioden auf den Leitungen platziert.

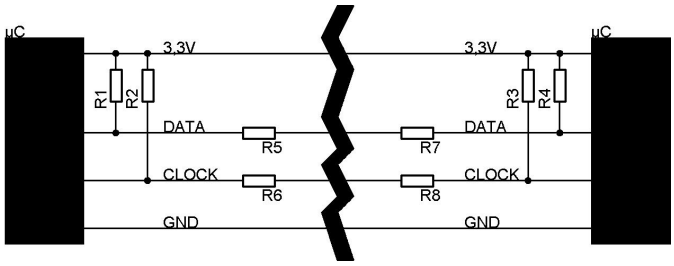


Abbildung 3.11: Schema der Kommunikationsverbindungen Box - Box TODO:+DIODEN

Der Leitungszustand befindet sich i.d.R. auf 1, falls kein Controller kommuniziert. Dieser Zustand wird erreicht, indem die beiden Controller an den Leitungsenden einen hohen Innenwiderstand zur Masse geschaltet haben. Dazu schaltet der Controller seinen Pin auf INPUT. Beide Controller haben jederzeit die Möglichkeit, die Leitungen auf den Zustand 0 zu setzen. Damit initiieren sie den Sendevorgang. Dies ist möglich, indem der jeweilige sendende Controller die Leitung auf Masse zieht. Dazu muss der Controller den Pin auf OUTPUT schalten und eine '0' ausgeben. In Abbildung 3.12 sind die Testergebnisse der Schaltung auf dem Oszillographen zu sehen. Verwendet wurden für R1-R4 4,7kΩ und für R5-R8 47Ω.

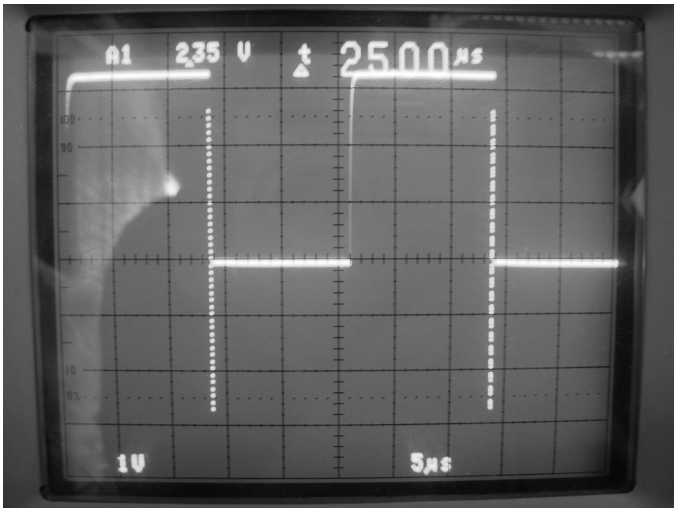


Abbildung 3.12: Ergebnisse des Oscillographentests

In Abb. 3.12 sind die beiden Leitungsstati zu sehen, die mithilfe der Schaltung aus Abb. 3.11 realisiert werden. Diese Zustände können nun zur Kommunikation zwischen den Spielfeldboxen genutzt werden.

Zur Realisierung der Kommunikation wird ein sehr einfaches Protokoll genutzt. Der sendende Controller legt das zu sendende Datum an der DATA-Leitung an und signalisiert daraufhin dem Nachbarspielfeld über die CLOCK-Leitung, dass ein neues Bit bereitsteht. Falls beide Controller gleichzeitig senden wollen, ensteht ein Fehler, da einerseits beide ihr eigenes Datum anlegen und andererseits evtl. keiner der Interruptaufforderung des Anderen nachkommt. Diese Kollisionswahrscheinlichkeit muss reduziert werden. Das kann dadurch erreicht werden, indem die verfügbare Bandbreite nur zu einem Teil ausgelastet wird. Die Frage bleibt dennoch offen, ob die Fehlerrate im realen Betrieb gering genug sein wird. Alternativ muss die Low-Level Kommunikation ein wenig angepasst werden. Im Test hat die Kommunikation jedoch einwandfrei funktioniert.

3.2.3 Spielfeldbox - Roboter

Die Kommunikation zwischen Spielfeld und Roboter lässt sich am besten mit LEDs in den einzelnen Feldern durchführen. Vorteil dabei ist, dass sichergestellt ist, dass der Roboter herausfinden kann, auf welchem Spielfeld er gerade steht. Bei einer Funkübertragung müsste die Ortung anders realisiert werden. Zur Auswahl stehen dabei LEDs im sichtbaren und LEDs im infraroten Bereich. Da die LEDs auf den Spielfeldern auch für die optische Darstellung von Aktionen (z.B. Laser) verwendet werden können, werden hier LEDs aus dem sichtbaren Bereich verwendet werden.

Im Folgenden wird das angewendete Kommunikationsprotokoll detailliert beschrieben.

Parameter

- Zielsendefrequenz: 4 kHz (nominal)
- absolutes Maximum Sendefrequenz: 5 kHz
- absloutes Minimum Sendefrequenz: 2 kHz
- absolutes Minimum Abtastrate: 8kHz (während aktiver Kommunikation)
- Roboter blinken (in Sendefrequenz):
  alternatives Senden/Lauschen-Verhältnis um Matching in 2 Zyklen zu garantieren
  - Einsen senden: 20 Takte
  - Nullen senden = lauschen: 3 Takte
- Spielfeld blinken (in Sendefrequenz):
  Hoher Anteil Einsen, damit Roboter sich an dem Licht ausrichten kann und es gegenüber Anwender als Dauereins erscheint.
  - Einsen senden: 38 Takte
  - Nullen senden = lauschen: 3 Takte

Idee
Die gerichtete Kommunikation besteht aus drei Phasen:

- Verbindungsaufbau
  Beide Kommunikationspartner stellen genügend Ressourcen zur Verfügung (insb. Rechenzeit) und stellen sich somit auf die Synchronisation und Kommunikation ein.
- Senden/Empfangen der Datenbits
  Die Datenbits werden jeweils zwischen einer Eins und einer Null übertragen, je 8 Bits pro Verbindung, der Sender kann den Wert per Majoritätsentscheidung ermitteln.
- Verbindungsabbau
  Eine abschließende Eins teilt das Ende des achten Datenbits (mit abschließender Null) mit.

Ausgangssituation
Der Roboter steht auf einem Spielfeld, der Roboter lauscht per Interrupt (ist i.d.R. in einem Sleepmodus) und das Spielfeld tastet den Fototransistor regelmässig ab.
Einer der Kommunikatinspartner möchte Daten senden (Roboter z.B. Fehlermeldung á la Batterie-ist-leer, Spielfeld z.B. Bewegungsbefehle). Dazu muss er selbst die Ressourcen für den Sendevorgang mit ausreichender Sendefrequenz bereitstellen. Der Kommunikationspartner muss informiert werden, damit er ebenfalls ausreichend Rechenressourcen für eine genügend hohe Empfangsabtastrate bereitstellen kann.

1. Verbindungsaufbau

- Sender: Reserviere Rechenzeit, beginne zu blinken und lauschen, bis (beim Lauschen) Eins beobachtet wird
- Empfänger (Roboter erwacht dank Interrupt, Spielfeld beobachtet ein Signal wegen gelegentlichen regelmäßigen Lauschens): Reserviere Rechenzeit, beginne zu blinken und lauschen, bis (beim Lauschen) zwei Nullen in Folge beobachtet werden.
- Sender: Schalte Blinken ab, lausche, bis zwei Nullen beobachtet werden
- Empfänger: Schalte Blinken ab, lausche [min 1/8 Sekunde, max 1/4 Sekunde] (siehe 2.)
- Sender: Beende lauschen, sende Daten (siehe 2.)

2a. Senden der Datenbits
Für alle acht Datenbits verfahre wie folgt (mit Sendefrequenz):

- Sende eine Eins
- Sende das Datenbit
- Sende eine Null

2b. Empfangen der Datenbits
Für alle acht Datenbits verfahre durch Lauschen (mit Abtastrate) wiefolgt:

- bestimme Dauer der Eins
- bestimme Dauer der Null
- bestimme per Majoritätsentscheidung über die Dauer der Einsen und Nullen das Datenbit

Dabei werden die Einsen und Nullen garantiert je durch eine positive bzw. negative Flanke beendet.

3. Verbindungsabbau

- Sender: Sende eine Eins, gib Ressourcen frei
- Empfänger: Verwerfe letzte Eins (wird nur wegen positiver Flanke benötigt), gib Ressourcen frei

Vorteile
Da gerade vom Spielfeld erwartet wird, mit vielen Kommunikationspartnern zu kommunizieren und die benachbarten Spielfelder per Interrupt angeschlossen sind, muss die Übertragung zwischen Spielfeld und Roboter unempfindlich gegenüber Sendefrequenzschwankungen sein. Dies soll über die Majoritätsentscheidung mit wesentlich höherer Abtastrate garantiert werden.

Einsatz des Protokolls

- Empfangsbestätigung per ID-Senden, im Fehlerfall sendet Roboter wiederholt Errormessages, Spielfeld muss Fehler oberhalb des Kommunikationsprotokolls behandeln, um nicht handlungsunfähig zu werden. Dies ist im Roboter nicht wichtig, da er nach nicht erfolgreicher Kommunikation sowiso handlungsunfähig ist.
- Dauert ein Wartevorgang länger als 1/8 Sek., dann beende Kommunikationsvorgang (vermeidet Deadlock, wenn ein Fehler auftritt)
- Wird eine Verbindung nicht erfolgreich durchgeführt, sollte mindestens 1/2 Sekunde bis zum nächsten Versuch gewartet werden, damit beide Kommunikationspartner sicher wieder für Kommunikation bereit sind.

3.3 Roboter

In diesem Kapitel werden die Anforderungen an die Spielfiguren in diesem Projekt, also die Roboter, erläutert. In Abschnitt 3.3.2 werden einige Standardlösungen vorgestellt, die zur Auswahl standen, und gleichzeitig werden die Entscheidungen bezüglich der einzelnen Komponenten begründet (vgl. 3.3.3). Die unternommenen Tests und der Aufbau des zugehörigen Testboards werden in Abschnitt 3.3.4 beschrieben. Der Abschnitt 3.3.5 über den mechanischen Aufbau wird sich damit befassen, welche Bauteile an welcher Stelle angebracht werden, und warum dies sinnvoll ist. Die Erläuterungen über den elektronischen Aufbau in Abschnitt 3.3.6 befassen sich unter anderem mit dem Layout der Platinen und der Anordnung der einzelnen Elemente. Kapitel 3.3.7 geht näher auf den Fehlerkorrekturalgorithmus ein, der es ermöglicht, die Bewegungen des Roboters gegebenenfalls zu berichtigen.

3.3.1 Anforderungen

Die Plastik- bzw. Zinnroboter des Robo-Rally-Spieles werden durch echte, funktionsfähige Roboter ersetzt. Dabei ist zunächst festzustellen, welche Fähigkeiten realisiert und welche Anforderungen erfüllt werden müssen. Diese ergeben sich aus den Anforderungen des Spieles sowie aus den Kompatibilitätsbedingungen bezüglich der Spielfeldboxen.
Die Roboter werden zwei Kernaufgaben haben:

1. Navigation:

   Jeder Roboter wird Spielzüge ausführen, d.h., er muss vorwärts und rückwärts fahrend das Spielfeld wechseln und sich auf der Stelle drehen können. Hierbei ist jeweils eine hohe Präzision notwendig [AMI99].

2. Kommunikation:

   Jeder Roboter wird mit dem Spielfeld interagieren müssen, das die Spielintelligenz ihm gegenüber repräsentiert. Diese wird das Spielgeschehen kontrollieren und den Roboter steuern. Hierzu gehören Fähigkeiten zur eigenen Identifikation, zur Annahme von Steuerkommandos sowie gegebenenfalls zur Sendung von Statusberichten. Die resultierende Kommunikation benötigt sowohl eine Sende- als auch eine Empfangseinheit (vgl. Kommunikationsprotokoll in Abschnitt 3.2.3).

Neben diesen Kernaufgaben gibt es eine Reihe von Einschränkungen, die beachtet werden müssen:

Die Größe ist durch die Spielfeldbox mit ihren quadratischen, 5 cm breiten Spielfeldern begrenzt (siehe 3.4). Da sich Roboter auf benachbarten Feldern ohne Behinderung und Berührung bewegen und insbesondere drehen können müssen, ist unter Berücksichtigung eines Sicherheitsabstandes ein Durchmesser vom maximal 4,5 cm realisierbar. Die Höhe sollte optischen und stabilisierenden Gesichtspunkten entsprechend durch etwa 6 cm begrenzt sein.

Ein Roboter muss über ausreichende Energiekapazität für mindestens ein Spiel verfügen, und somit, Erfahrungswerten folgend, für bis zu drei Stunden Betrieb gerüstet sein. Umweltpolitischen und ökonomischen Gesichtspunkten gerecht werdend sollte die Energiequelle kostengünstig in der Anschaffung und vor allem im Betrieb sein. Das heißt, sie sollte kostengünstig regenerierbar sein, um Verbrauchsmaterial bzw. -abfälle sowie Verbrauchskosten gering zu halten.

Ferner ist Robustheit gegenüber kleinen Schlägen und Stößen, Unebenheiten in der Fahrbahn und gegenüber Staub notwendig, um den Spielbetrieb und eine akzeptable Roboterlebenserwartung zu sichern.

Unter Berücksichtigung dieser Aufgaben und Einschränkungen gilt es, Standardtechniken der Robotik zu bewerten und gegebenenfalls eigene Lösungen zu entwickeln.

3.3.2 Standardlösungen

Zu recht günstigen Preisen sind eine Vielzahl von Mikrocontrollern, Sensoren und Aktuatoren im Elekronikhandel erhältlich. Durch eine fast unbegrenzte Anzahl an Kombinationsmöglichkeiten sind der Kreativität für Roboter bezüglich ihrer Hardwareausstattung, aber auch ihrer zu realisierenden Fähigkeiten kaum Grenzen gesetzt. So haben sich beispielsweise große Interessensgemeinschaften im Internet gebildet [Fra06], um ihre Lösungen und Erfahrungen auszutauschen. Aber auch die Industrie hat reagiert und vorgefertigte Bausätze entworfen [BSW06].

Es entstanden so eine Reihe von Standardlösungen, die hier zu betrachten und bezüglich ihrer Verwendbarkeit für die Spielroboter zu bewerten sind. Dabei ist insbesondere die

Größe und das Gewicht der Komponenten kritisch zu untersuchen, denn Standardtechniken sind zum Großteil auf wesentlich größere Roboter (bis 20 kg) ausgerichtet.

3.3.2.1 Fortbewegung

Zwei Arten der Roboterfortbewegung dominieren den Markt: Fahren (Radantrieb, Kettenantrieb) und Laufen [BSW06]. Dabei ist ein fahrender Roboter wegen der einfacheren Mechanik und Programmierung leichter zu realisieren. Aufgrund der strikten Platzbegrenzung sind Radantriebe zu bevorzugen.

Diese sind am Weitesten verbreitet und lassen sich in mehrere Kategorien klassifizieren [B$^+$06c]. Wichtiges Kriterium ist hierbei die Anzahl der Antriebsräder.

Ein Antriebsrad (analog zum Fahrrad oder Motorrad) macht eine Lenkung erforderlich. Da zudem keine Drehung auf einem Punkt möglich ist, sind Antriebe dieser Kategorie für die Spielroboter ungeeignet.

Bei zwei Antriebsrädern sind Varianten mit oder ohne Lenkung denkbar, und je nach Positionierung der Antriebsräder gibt es auch Lösungen, die eine Drehung auf einem Punkt ermöglichen. Hierzu müssen beide Räder an der Mittelachse des Roboters angebracht sein. Dieser darf keine weiteren richtungsbeeinflussenden Komponenten haben wie beispielsweise passive Räder eines Pkw, sondern lediglich Gleiter oder frei drehbare Stützräder. Da letztere zu Irritationen in der Bewegung des Roboters führen können, ähnlich den frei drehbaren Vorderrädern eines Einkaufswagens, sind Gleiter zu bevorzugen.

Drei oder mehr Antriebsräder können allein aus Gründen mangelnder Raumkapazität und begrenzter Energiereserven nicht zum Einsatz kommen.

Ein Roboter basierend auf einer mittelachsangetriebenen, zweimotorigen Fortbewegung mit Stützgleitern vorne und hinten erscheint viel versprechend. Dabei sind zwei Motoren notwendig, da die Antriebsräder sowohl in die gleiche (zum Vorwärts- und Rückwärtsfahren) als auch in unterschiedliche Richtungen (zum Links- sowie Rechtsherumdrehen auf einem Punkt) laufen können müssen. Eine einmotorige, getriebegestützte Lösung fand bisher keinen Einzug in die Standards der Robotik.

3.3.2.2 Motoren

Insbesondere für die Auswahl der Motoren gilt, dass Größe, Strombedarf und Steuerbarkeit zu beachten sind. Dabei impliziert die Steuerbarkeit, dass sich die Motoren durch klare Schrittweiten oder präzise Geschwindigkeiten auszeichnen müssen, um zurückgelegte Wege genau zu bestimmen sowie ein gerades Fahren erst zu ermöglichen. Unterschiedliche Motoren ließen den Roboter einen Bogen fahren. Des Weiteren muss ein präzises Halten der Roboter (Brems- und Haltefunktion) möglich sein.

Weitere technische Aspekte wie das Drehmoment der Motoren sind sekundär, da die Roboter nur wenige hundert Gramm wiegen werden und durch nahezu alle verfügbaren Motoren fortbewegt werden können.

Für fahrende Roboter werden meist (Gleichstrom-)Getriebemotoren eingesetzt [GG04], vereinzelt aber auch Schrittmotoren, zumeist in Form modifizierter Servomotoren[1] [Par]. Servomotoren selbst können sich allerdings nicht beliebig weit drehen und finden eher in laufenden Robotern Einsatz [Lyn] und sind daher hier unbedeutend. Getriebemotoren und Schrittmotoren sind aber näher zu betrachten.

Getriebemotoren

Getriebemotoren [B$^+$06b] zählen zu den einfachsten, kleinsten und universellsten Motoren. Durch unterschiedliche Untersetzungen eignen sie sich für schnelle aber auch für kraftvolle Einsätze. Sie werden zum Beispiel für das Öffnen und Schließen von DVD - Laufwerken benutzt.

Aufgrund der geringen Größe erscheint ein Einsatz dieser Motoren in den Robotern des Projektes interessant. So sind 1,8 g schwere Motoren mit 6 mm Durchmesser und 15 mm Länge erhältlich [Mik03b].

Darüber hinaus ist die Motoransteuerung einfach. So gibt es lediglich zwei Anschlüsse, die mit unterschiedlicher Polarität zu belegen sind, um den Motor laufen zu lassen. Die Richtung des Stromflusses ist ausschlaggebend für die Drehrichtung des Motors. Da diese im Betrieb wechseln können soll und ein Motor zudem mehr Strom benötigt als ein Mikrocontrollerausgang in der Regel liefern kann, ist eine Ansteuerung über einen Motortreiber notwendig [Alt98], im einfachsten Fall über eine H-Brücke[2] [GG04].

In der Regel kommt ein IC der Serie L293 [Tex04] in Robotern zum Einsatz. Dabei steht je H-Brücke[3] ein Eingang (Enable) zur Verfügung, der den zugehörigen Motor ein- und ausschaltet. Dieser kann per PWM Signal[4] so betrieben werden, dass die Geschwindigkeit einigermaßen kontrollierbar ist, was bei einem Leerlauf von oft über 10.000 Umdrehungen pro Minute erforderlich ist. Zwei weitere Eingänge bestimmen über ihre Signalbelegung (low oder high) die Drehrichtung des Motors.

Derartige Motortreiber bieten zugleich die Möglichkeit, Brems- und Haltewirkungen zu realisieren. Durch Belegung der Eingänge mit gleichen Pegeln wird der Motor kurzgeschlossen, was zu einer Bremswirkung führt, ohne die die Getriebemotoren nachlaufen

[1] Servomotoren enthalten zumeist eine mechanische Sperre, die den Drehwinkel der Motorachse einschränkt. Diese kann nach Aufbrechen der Motorengehäuse zumeist gewaltsam entfernt werden. Bei der von uns anvisierten Größenordnung von einigen Millimetern Durchmesser ist dies jedoch nicht empfehlenswert.

[2] Eine H-Brücke besteht im einfachsten Fall aus vier Transistoren, eine genauere Betrachtung ist in Abschnitt 3.3.6.5 zu finden

[3] Einige Ausführungen der L 293 Serie erhalten mehrere H-Brücken.

[4] Pulsweitenmodulation

würden. Dies ist auch bei Richtungsänderung der Motorlaufrichtung zwischenzeitlich empfehlenswert, um die Belastung der Motorsteuerung und des Motors selbst möglichst gering zu halten.

Getriebemotoren können, abhängig von ihrer Belastung, dem Untergrund und individuellen Produktionsunterschieden, in ihrer tatsächlichen Umlaufgeschwindigkeit voneinander abweichen. Ob zwei von verschiedenen Motoren angetriebene Räder sich gleich schnell beziehungsweise gleich oft gedreht haben, kann ohne externe Sensoren, die die Radumdrehungen überwachen, nicht kontrolliert werden.

Für das R_3D_3 Projekt scheinen Getriebemotoren daher unpraktikabel zu sein, denn schon die Fahrt gerader Strecken würde zur Herausforderung werden. Eine Haltewirkung wäre zudem nur durch einen Kurzschluss mit entsprechendem Stromverbrauch möglich.

Schrittmotoren

Schrittmotoren sind grundlegend anders konzipiert als Getriebemotoren [Bac03]. Ein Rotor (oft ein Permanentmagnet) wird durch spulenerzeugte, elektrische Magnetfelder bewegt. Dabei wechseln die Spulenenden ihre Polarität durch unterschiedliche Belegung der Spulenanschlüsse, so dass sich der Rotor Stück für Stück im Kreis fortbewegt.

Die Fortbewegung des Rotors pro Schritt ist relativ präzise (in der Regel höchstens 5% Toleranz), pro vollständiger Drehung sogar genau festgelegt, denn produktionsbedingte individuelle Abweichungen werden innerhalb einer vollständigen Rotordrehung wieder ausgeglichen, da die Position der Spulen fest ist. Folglich eignen sich Schrittmotoren für präzise Ansteuerungen.
Dabei liegt die Schrittgröße, je nach Bauart des Motors, in der Regel zwischen 1,8 Grad und 18 Grad, kann aber über Getriebe nach Belieben verkleinert werden. So ist eine hohe Präzision und Gleichläufigkeit verschiedener Motoren erreichbar.

Das Haltemoment eines Motors ist etwa gleich der Größe des Drehmomentes. Da zudem die zurückzulegende Strecke durch eine entsprechende Motordrehschrittanzahl zu bestimmen ist, bedarf es keiner Bremse.

Seit Jahren wird kontinuierlich die Größe von Schrittmotoren minimiert, und so sind mittlerweile sehr kleine Vertreter erhältlich, zum Beispiel mit einem Durchmesser von 6 mm und einer Länge von 10 mm zuzüglich einer Antriebswelle bei 10 g Gewicht [Nan04a].

Im Vergleich zu den Getriebemotoren ist jedoch die Ansteuerung der Schrittmotoren aufwendiger. Es ist notwendig, an jedem Spulenende eine individuell zu bestimmende Polarität anlegen zu können. So müssen pro Motor mindestens vier Anschlüsse belegt werden. Damit der Rotor sich schließlich Schritt für Schritt fortbewegt, sind zudem spezielle Folgen von Polaritätsmustern abzuarbeiten.

Da aber die Motoren über einen Mikrocontroller anzusteuern sind, können die entsprechenden Polaritätsmusterfolgen leicht als Bitmusterfolgen realisiert werden. Dabei läuft

die Ansteuerung wie bei Getriebemotoren über Motortreiber, wobei sich lediglich die Anzahl der benötigten Bausteine vergrößert.
Der Enable-Eingang ist nun nicht mehr mit einem PWM Signal zu betreiben, sondern er dient lediglich dem Ein- und Ausschalten der Motoren. Die Geschwindigkeit der Motordrehung wird über die der Änderungen der Bit- bzw. Polaritätsmuster bestimmt.

Schrittmotoren bieten also für präzise Steuerungen wesentlich bessere Möglichkeiten als Getriebemotoren.

3.3.2.3 Mikrocontroller

Die Steuerung eines Roboters wird von einem Mikrocontroller realisiert. Eine Auswahl interessanter Mikrocontroller ist im Abschnitt 3.1 zu finden. Hier folgt eine Analyse der in Robotern üblicherweise verwendeten Controller.

Bei den Bausätzen und fertigen Robotern dominieren zwei Prozessorfamilien den Markt. Im unteren Preissegment werden Prozessoren der ATMega-Familie von Atmel eingesetzt (z.B. im Asuro oder Crash Bobby [BSW06]). Dabei handelt es sich um kostengünstige 8-Bit RISC-Prozessoren[5], die für einfache Anwendungen genügend Ausstattung bieten.
Im oberen Preissegment kommen Motorola-Prozessoren der 68000 Reihe zum Einsatz (z.B. im Soccer eyeBot oder Khepera II [BSW06]). Dabei handelt es sich um 32-Bit Prozessoren, die einerseits eine gute Ausstattung, beispielsweise größere Speicher, bieten, andererseits jedoch erheblich teurer sind als Atmels ATMega-Prozessoren.

Während die Atmel-Prozessoren sich eventuell für einen Einsatz in den Robotern eignen, sind die Prozessoren der 68000 Reihe von Motorola für den Einsatz in dieser Projektgruppe zu teuer.

3.3.2.4 Stromversorgung

Abhängig von den verschiedenen Komponenten wie Sensoren, Aktuatoren (insbesondere die Motoren) und Mikrocontrollern haben sowohl die Roboterbausätze als auch die Eigenbauten einen unterschiedlichen Strombedarf. Entsprechend gibt es eine große Anzahl verschiedener Energiequellen, die sich in Spannung, Kapazität und Energiedichte[6] unterscheiden. In der Praxis haben sich Batterie- bzw. Akkulösungen durchgesetzt. In der Robotik sind alle verbreiteten Batterie- und Akkustandards vertreten wie zum Beispiel Nickel-Metall Hydrid (NiMH) oder Lithium-Ionen (Li-Ion) Lösungen.

Da die Spielroboter des Projektes langlebig und klein sein sollen, sind Energiequellen mit hoher Energiedichte zu bevorzugen. In Anbetracht des Kosten- und Umweltbewusstseins erscheinen Akkus viel versprechend. Eine Auswahl kann aber erst getroffen werden, wenn

[5] Der ATmega8 ist zurzeit bei Farnell für deutlich unter 10 Euro erhältlich.
[6] Energiekapazität pro Volumeneinheit

die zu versorgenden Komponenten mit ihrem Bedarf an Spannung und Leistung bekannt sind.

3.3.2.5 Gehäuse und Design

Dem Design von Robotern scheinen keine Grenzen gesetzt zu sein: neben unscheinbaren rollenden Zylindern [KTe] sind alle Formen der Natur Vorbild. So gibt es Spinnen [Lyn], Hunde [Son] und dem Menschen nachempfundene Roboter [Hit]. Insbesondere Roboterhunde haben einen hohen Bekanntheitsgrad erworben [Red04].

Aufgrund der stark einschränkenden Bedingungen werden derart aufwendige Designs in diesem Projekt keine Rolle spielen. Es sind vielmehr praktische, klare und einfache Formen vorzuziehen.

3.3.2.6 Fazit

Es gibt zwar eine Reihe von Roboterimplementierungen, in denen viele interessante Techniken zum Einsatz kommen, aber keiner dieser Standardroboter entspricht annäherungsweise den Spezifikationen und Anforderungen unseres Projektes. Daraus folgt, dass sich keine Lösung direkt auf die Projektroboter anwenden lässt.

Vielmehr sind die meisten Roboter erheblich größer, dafür aber auch flexibler einsetzbar. Entsprechend sind viele Lösungsansätze unpraktikabel, da sie zu groß, funktional überdimensioniert oder den Anforderungen nicht gewachsen sind.

So gilt es, Standardlösungsansätze zu präzisieren, zu modifizieren, echte Alternativen aufzuzeigen, sie bezüglich unserer Applikation zu beurteilen und so ein stimmiges Gesamtkonzept zu entwickeln.

3.3.3 Komponentenauswahl

Aufbauend auf die in 3.3.2 vorgestellten Standardlösungen werden die einzelnen Komponenten der Roboter ausgewählt. Dazu wird hier auf Motor (3.3.3.1) und Motortreiber (3.3.6.5), Getriebe (3.3.3.3) und Räder (3.3.3.4) sowie letztendlich auf das Gehäuse (3.3.3.5) eingegangen, bevor der Energieverbrauch analysiert werden kann, um eine passende Energiequelle (3.3.3.6) auszuwählen.

Die Auswahl des Mikrocontrollers AT91SAM7S32 ist in 3.1.3.1 zu finden.

3.3.3.1 Motor

Wie bereits in 3.3.2.2 beschrieben, sind für den Roboter Schrittmotoren am Besten geeignet. Da diese sehr klein sein müssen, ist die Auswahl an Herstellern stark eingeschränkt.

Die Motoren von Nanotec [Nan] und Faulhaber [Fau] sind verbreitet und entsprechen den Größenanforderungen des Projektes. Da jedoch Faulhaber primär Servomotoren vertreibt, sind Lösungen von Nanotec einzusetzen.

Lediglich Motoren der SP-Serie[7] kommen aufgrund ihrer Größe in Frage, hierbei insbesondere der SP0618M0104 [Nan04a] und der SP1018M0204 [Nan04b]. Sie unterscheiden sich in ihrer Größe (6 mm Durchmesser gegenüber 10 mm) und ihrer Leistungsaufnahme (0,14 A pro Wicklung bei 1,96 V gegenüber 0,22 A pro Wicklung bei 3,3 V).
Der kleinere SP0618M0104 wird vorgezogen, um Platz und Energie einzusparen.

3.3.3.2 Motortreiber

Im Gegensatz zu den vom Mikrocontroller gesendeten Steuersignalen unterscheidet sich die vorgeschaltete Ansteuerungslogik eines Schrittmotors nicht wesentlich von der eines Gleichstrommotors. Insofern wird im Folgenden erst auf die Notwendigkeit einer Motorsteuerung allgemein eingegangen. Daran anschließend werden unter den gegebenen Eigenschaften des in 3.3.3.1gewählten Motors neben einigen Standardbausteinen auch eigene Konstruktionsmöglichkeiten betrachtet, bevor schließlich die Auswahl des Treibers begründet wird.

Notwendigkeit eines Motortreibers

Die geforderte Leistung eines Motors kann in fast allen Fällen nicht direkt vom Mikrocontroller abgegriffen werden, da dieser unter der Last zerstört würde. Deswegen ist eine Zusatzschaltung zur Trennung notwendig.
Der einfachste Weg, eine Leitung vom Mikrocontroller abzukoppeln ist, mit dieser einen Transistor anzusteuern, der dann den größeren Strom zur Verfügung stellt.
Da die Polung des Motors je nach Drehrichtung geändert werden muss, genügt diese einfache Schaltung jedoch nicht, sondern muss entsprechend erweitert werden. Die einfachste Möglichkeit ist eine Schaltung aus 4 Transistoren, die sogenannte H-Brücke(vgl. [DPR]). Der Name entspringt der im Bild 3.13 erkennbaren H-förmigen Anordnung der Transistoren auf dem Schaltplan.

Die H-Brücke erlaubt es, den Motor in drei definierte Zustände zu versetzen.

Zu diesen Zuständen gibt es jedoch noch einen weiteren, den Kurzschluss. Werden etwa A_1 und B_2 geschaltet, werden Spannungsversorgung und Masse verbunden. Dieser Zustand kann durch eine kleine Logikschaltung verhindert werden. Werden die zwei Transistoren einer Seite durch jeweils einen Inverter getrennt an eine Leitung gelegt, erhält man zwei Steuerleitungen und eliminiert die Kurzschlussgefahr. Die Inverter kann man sich

[7] Die SP-Serie sind Schrittmotoren, die einen Permanentmagneten als Rotor nutzen.

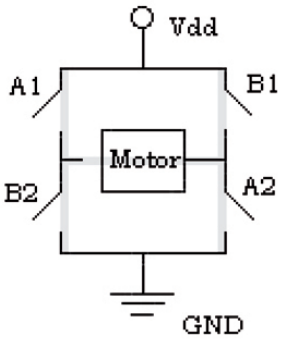


Abbildung 3.13: Schematische Darstellung einer H-Brücke

hier sparen, wenn man bei der Programmierung darauf bedacht ist, diesen Fall logisch auszuschließen.

Der bauliche Unterschied bei Schrittmotoren kommt nun nur in dem Aspekt zum Tragen, dass bei den Schrittmotoren mehrere Spulen verbaut sind (vgl. Abb. 3.14), die jeweils durch eine H-Brücke angesteuert werden müssen. Alle weiteren Veränderungen wirken sich nur auf die Programmierung (3.3.7.6) der Signale vom Mikrocontroller aus.

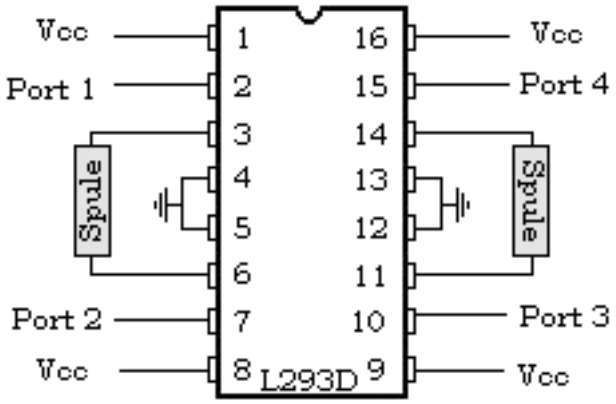


Abbildung 3.14: Anschluss eines Schrittmotors am L293D (Quelle: [B+06a])

Konstruktion eines Motortreibers

Der Aufbau einer H-Brücke wurde bereits im vorigen Abschnitt erläutert. Diese bildet im Motortreiber die Hauptkomponente. Die notwendige Schaltung wird jedoch noch etwas komplexer, da durch die Spulen der Motoren beim Abschalten Induktionsspannungsspitzen beträchtlicher Höhe entstehen können (vgl. [McM04]). Elektrotechnisch gesehen

gleicht der Motor in diesem Moment, im rotierenden Zustand, einem kleinen Stromgenerator. Damit diese Ströme nicht zum Mikrocontroller gelangen, sollte er durch zwei am Motor in Sperrichtung installierten Dioden abgeschirmt werden.
Ein problematisches Kriterium bei der Eigenkonstruktion ist also der Platzbedarf, der sich selbst mit Tricks (vgl. [vV]) aufgrund der Größe der einzelnen Komponenten trotz großer Mühen nur bedingt minimieren lässt.
Auch Hybridlösungen sind denkbar. Es gibt reine H-Brücken, wie die ZXMHC6A07T8 [Zet04], die nur aus den Transistoren bestehen, diese jedoch auf wesentlich kleinerem Platz als Einzelbausteinen unterbringen. Hier müssten dann wenigstens die Dioden nachgerüstet werden.
Ein weiterer Lösungsansatz liegt in der Nutzung von Logik-Bausteinen, wie den 74x245, die hierfür zweckentfremdet werden können (vgl. [Pri]). Dabei ist dann darauf zu achten, dass die Leistungsgrenzen der Bausteine nicht (dauerhaft) überschritten werden.
Insofern kann der Eigenbau in Größe, Preis und Funktionssicherheit nicht mit den fertigen Bausteinen mithalten, die zudem meist zusätzlich noch mindestens einen Thermalschutz beinhalten.

Fertige Treiberbausteine

Eine sehr zahlreiche Auswahl an Motortreibern ist von verschiedenen Chipherstellern als IC verfügbar. Auch Nanotec, der Anbieter der in 3.3.3.1 ausgewählten Motoren, betreibt zwei Serien von Treibern (SMC und IMT90x [Nan]), die zusätzlich noch Mikroschrittfähig sind. Das heißt, sie können bis zu $\frac{1}{16}$ große Teilschritte machen und so teilweise ein Getriebe ersetzen. Diese haben jedoch vor allem zu hohe Anforderungen an die Spannungsversorgung und sind deswegen für diesen Aufbau ungeeignet.
Im Eigenbau gern genutzte Standardlösungen wie der TI L293[Tex04] oder der ST L298 [STM00] leiden unter dieser Vorgabe, wie auch viele andere Treiber, die erst ab 4.5 V ihren Dienst aufnehmen (beispielsweise TC 4469, [Tel96]).
Eine andere große Gruppe, die schließlich auch mit Besonderheiten wie größeren Thermal- und Shutdownlogiken erweitert sind, disqualifiziert sich durch ihre Dimension (beispielsweise LMD 18200, [Nat]).

HIP4020

Der HIP4020 [Int05] von Intersil ist ein Motortreiber, der mit 3,3 V zurecht kommt und zudem in SMD Bauform mit 13 x 10 x 2.35 mm recht klein, aber immerhin größer als der Mikrocontroller ist. Mit einer Schaltkapazität von 500 mA, einem Leerlaufstrom von 0,8 mA und einer guten Verfügbarkeit genügt er den definierten Ansprüchen. Ein weiterer Pluspunkt sind die Erfahrungen einer früheren Projektgruppe, die sowohl diesen als auch selbst gebaute Treiber benutzt haben.
Aufgrund von zwei Spulen pro Motor sind hier für die zwei Mototren also vier HIP4020 Bausteine als Treiberstufe notwendig.

Kompatibilität Da die Motoren eine Eingangsspannung von 1,96 V erwarten, der HIP4020 jedoch 3,3 V liefert, muss eine Spannungsanpassung stattfinden. Dazu muss ein Widerstand als vorgeschaltete Spannungsreduktionseinheit eingefügt werden. Die Berechnung dessen Größe wird im Detail in 3.3.6.4 erläutert.

Testschaltung Um diese Kompabilität sicherzustellen, wurde eine kurze Testschaltung gebaut. Dazu genügt eine kleine SMD Platine, auf der der HIP4020 befestigt und die entsprechende Verdrahtung mit angelöteten Kabeln durchgeführt werden. Bei diesem Test lässt sich feststellen, dass der Baustein zuerst in einem undefinierten Zustand ist, dabei jedoch den Motor voll belastet. Als direkte Folge erwärmt sich der Motor stark, was zu vermeiden ist. Es müssen also später bei der Programmierung (vgl. 3.3.7.6) entsprechende Vorkehrungen getroffen werden, diesen undefinierten Zustand schnellstmöglichst zu verlassen.

Von diesen zwei Punkten abgesehen, erfüllt der HIP4020 die gestellten Anforderungen genau. Ein weiterer Baustein, der LB1836M [SAN], scheint dies ebenso zu tun, beinhaltet jedoch bei kleinerer Größe (8 x 6.4 x 1.8) gleich zwei H-Brücken. Für die Nutzung müsste das Layout der Platine nur leicht geändert und gegebenenfalls die Lage des Widerstandes derart korrigiert werden, dass er weiter vom Motor entfernt positioniert ist. Dieser Baustein scheint jedoch zumindest in Deutschland nicht erhältlich zu sein.

3.3.3.3 Getriebe

Die Umsetzung der Motorkraft auf die Antriebsräder wird im Roboter durch Getriebe realisiert. Die Notwendigkeit von Getrieben ergibt sich aus mehreren Anforderungen:

- Da die in 3.3.3.1 gewählten Schrittmotoren einen Schrittwinkel von 18° haben, wäre die Bewegung der Roboter unpräzise, das heißt eine minimale Änderung der Position des Roboters durch einen Schritt des Motors wäre relativ groß. Um jedoch die im Spiel notwenige Genauigkeit zu erreichen, sind feinere Bewegungen nötig. Dies wird mit einer Untersetzung eines Getriebe erreicht. Diese Untersetzung sollte mindestens 20:1 betragen, um den Spielanforderungen gerecht zu werden. Um aber eine brauchbare Bewegungsgeschwindigkeit des Roboters sicher zu stellen, sollte sie auch nicht über 100:1 liegen.

- Zudem wird durch die Reduktion der Drehzahl das am Rad einwirkende Drehmoment vergrößert. Dies führt bei gleich bleibender Leistung zu einer Lastreduktion am Motor, der die benötigte Kraft nun durch Drehzahl aufbringen kann und so geschont wird.

Getriebealternativen

Grundsätzlich stehen zwei Alternativen zur Verfügung: fertige Getriebelösungen können bezogen oder eigene Lösungen über Zahnräder und Schnecken zusammengebaut werden.

Vor allem beim Bezug von Getrieben gibt es eine große Hightech-Auswahl (insbesondere z.B. bei Faulhaber [Fau]):

- Planetengetriebe:
  Diese Lösung basiert auf Zahnrädern, die sich im Getriebe selbst mitdrehen. Auf diese Weise werden sehr hohe Untersetzungen (1000:1 und mehr) auf kleinem Raum erreicht (Cent-Stück-Größe).
- Stirnradgetriebe:
  Hierbei handelt es sich ebenfalls um eine zahnradbasierte Lösung, bei der sich allerdings die einzelnen Zahnräder nur um eine feste Achse drehen. Sie zeichnen sich insbesondere durch extrem hohe Robustheit aus, können beispielsweise bei verschiedensten Temperaturen und im Vakuum betrieben werden.
- Schneckengetriebe:
  Diese Getriebe nutzen eine Schnecke auf der Motorachse und werden in der Regel durch weitere Zahnräder ergänzt. Schneckengetriebe werden insbesondere im Hobbybastlerbereich genutzt, da sie vergleichsweise einfach zusammen zu bauen sind.

Da Hightech-Planeten- und Stirnradgetriebe sehr teuer sind, verbleiben für das Projekt lediglich Schneckengetriebe, die in Hobbybastlerläden wie Mikroantriebe [Mik05] vertrieben werden.

Eigenbaulösungen würden den Schneckengetrieben sehr ähneln: von einer Schnecke auf der Motorachse angetrieben, müsste im einfachsten Fall direkt ein Zahnrad auf der Radachse sitzen. Um höhere Untersetzungen zu realisieren, wären allerdings noch Zahnräder auf zusätzlichen Achsen einzusetzen. Da Eigenbaulösungen jedoch recht empfindlich und fehleranfällig sind, schließlich ist mechanische Genauigkeit nicht nur zuletzt wegen der Laufruhe gefordert, sind fertige Getriebe vorzuziehen.

Getriebeauswahl

Alle Getriebevarianten sind vom Konzept her geeignet, um im Roboter eingesetzt zu werden, da dieser außer einer Zieluntersetzung zwischen etwa 20:1 und 100:1 kaum Anforderungen an diese stellt. Daher rücken eher sekundäre Merkmale in den Vordergrund: der Preis und die Verfügbarkeit. Vor allem letztere erweist sich als Problem, da zum Beispiel Faulhaber auf Geschäftskunden mit entsprechendem Abnahmevolumen abzielt und deren Produkte daher primär in fertigen Robotern zum Einsatz kommen (z.B. im c't Roboter [BTT06]).

Als günstige, gut verfügbare Lösung erweisen sich die Getriebebausätze der Firma Mikroantriebe. Insbesondere das Bausatzgetriebe G30 [Mik03a] scheint geeignet, da es mit 30:1 der Zielübersetzung entspricht und zudem extra für Motoren mit 6 mm Durchmesser konzipiert wurde. Es besteht aus einer simplen, zweistufigen Zahnradkonstruktion, die vom Motor über eine Schnecke mit einem Innendurchmesser von 0,8 mm angetrieben wird und die Kraft auf eine 1 mm starke Achse überträgt, auf der dann das Rad befestigt wird. Da diese Lösung vergleichsweise günstig ist, wird diese in den Robotern verwendet.

3.3.3.4 Räder

Die Auswahl der Räder ist durch ihre maximale Größe der Roboter und deren Materialbeschaffenheit begrenzt. Um auf dem Spielfeld wenden zu können und um die Räder ziemlich weit außen platzieren zu können, ist ein Durchmesser von circa einem Zentimeter angebracht. Die Räder sollten gummiert sein, um die besten Voraussetzungen mit zu bringen, den Roboter auf der Spielfeldoberfläche fahren lassen zu können. Die Achsen müssen einen Durchmesser von einem Millimeter haben, um in die Getriebe zu passen (vgl. 3.3.3.3).

Somit stehen Räder zur Auswahl, die im Modellbaubedarf zu erwerben sind. LKW Räder im Maßstab 1:87 [HER06] entsprechen den Anforderungen. Bei Tests, die mit dem lauffähigen Roboterprototypen unternommen wurden, ergab sich jedoch, dass die Räder nicht in der Lage waren, Grip auf dem Makrolon zu entwickeln (vgl. 3.5.1.1). Um diesem Problem kurzfristig entgegen zu treten, werden die Reifen mit Haarspray besprüht. Auf lange Sicht gesehen stellt das jedoch keine Lösung dar, da die Räder ständig erneut behandelt werden müssten. Eine Alternative besteht darin, die Räder in Eigenarbeit zu konstruieren. Dazu gibt es bereits einen ersten Prototypen. Dabei werden Achse und Felge aus einem Stück Metall gedreht. Auf die Felge wird dann ein breiter Dichtungsring aus Gummi, erhältlich im Baumarkt, gezogen. Um die Getriebe weiter verwenden zu können, muss bei der Herstellung darauf geachtet werden, dass der benötigte Achsdurchmesser erreicht wird.

3.3.3.5 Gehäuse

Zu diesem Zeitpunkt existiert noch kein Gehäuse für den Roboterprototypen. Wie in Abschnitt 3.3.2.5 schon erwähnt, wird die Wahl wohl auf ein einfaches Design fallen, das im Sommersemester festgelegt wird.
Zur Befestigung des Gehäuses an der Bodenplatte gibt es bereits mehrere Lösungsansätze. Eine Möglichkeit besteht darin, Gewinde in beide Teile zu schneiden, und diese dann miteinander zu verschrauben. Des Weiteren könnte das Gehäuse von außen mit drei Schrauben an der Bodenplatte befestigt werden. Detailliertere Ausführungen ergeben sich dann im Sommersemester.

3.3.3.6 Energie-Versorgung

Wie in 3.3.2.4 beschrieben, sind zunächst der Bedarf an Spannung und Strom zu analysieren.

Bedarfsanalyse

Den höchsten Spannungsbedarf im Roboter hat der Mikrocontroller AT91SAM7S32, der 3,3 V benötigt [Atm05c], die dementsprechend bereit zu stellen sind.
Da die Spannung konstant sein muss, ist sie über einen Festspannungsregler abzusichern, der selbst eine höhere Eingangsspannung benötigt. Beim KF33BD [STM04] beträgt die Dropoutspannung[8] etwa 0,3 V, so dass die Energiequelle insgesamt 3,6 V zur Verfügung stellen muss.

Der Strombedarf des Roboters ergibt sich aus der Summe des Strombedarfs der einzelnen Komponenten. Die Komponenten mit dem höchsten Energieverbrauch sind die Motoren und der Mikrocontroller. Die beiden Motoren brauchen bei voller Last je bis zu 280 mA, der Mikrocontroller bis zu 50 mA. Es werden also grob abgeschätzt etwa 650 mA benötigt. Praktische Tests haben jedoch ergeben, dass bei Betrieb mit Motoren etwa 500 mA, ohne Motoren gar weniger als 100 mA Strom verbraucht werden. Eine Kapazität von etwa 1000 mAh wird für ein Spiel ausreichen, da der Roboter regelmäßig in Energiesparmodi versetzt werden kann.

Die Bauform des Roboters beschränkt die Größe der Energiequelle: bei einem Aussendurchmesser von 45 mm und einer Höhe von etwa 60 mm darf die Energiequelle maximal einen Durchmesser von 44 mm besitzen (z.B. 40 mm x 5 mm bei quaderförmiger Bauweise) und maximal 42 mm hoch sein[9].

Anforderungen an die Energiequelle

Wie schon in 3.3.2.4 beschrieben, kommen für die Roboter nur Akkus in Frage. Ein weiterer wichtiger Ausschlussgrund für Batterien besteht darin, dass diese regelmäßig auszuwechseln sind und dementsprechend die Batterie im Roboter gut zugänglich sein muss. Dies ist bei den gegebenen Rahmenbedingungen schwer realisierbar.

Die Bedarfsanalyse zeigt, dass die verwendeten Akkus sehr klein sein, gleichzeitig aber hohe Spannung und Energiekapazität aufbringen müssen. Daher kommen Lithium - Akkus (Lithium Ionen oder Lithium Polymer Akkus) mit einer hohen Energiedichte in die engere Auswahl. Diese sind als Handy- und Modellbauakkus erhältlich.

[8] Die Dropoutspannung ist die Differenz zwischen Eingangs- und Ausgangsspannung, die der Festspannungsregler benötigt, um seiner Aufgabe nachzukommen.
[9] Etwa 18 mm Höhe werden im Roboter durch Bodenabstand, Gehäuse und Kommunikationseinheit am Boden benötigt.

Handyakkus sind für die Verwendung in unseren Robotern nicht geeignet, weil sie von der Bauform her speziell für die Handyform konzipiert sind. Zudem sind sie weitaus teurer als Modellbauakkus.

Ein Lithium Polymer Akku [Wes04] mit günstiger Bauform, sehr hoher Energiedichte und einer passenden Spannung von 4,2 V ist der Kok 1200 von Kokam. Lediglich die Höhe mit 52 mm ist größer als erwünscht, Alternativen hätten dieses Kriterium allerdings stärker verletzt. Die Roboterhöhe steigt somit auf knapp 7 cm, die weitere Konzeption des Roboters wird dadurch jedoch nicht beeinträchtigt.

Über das Ladegerät ULTRAMAT 12 [Gra04] ist eine Ladung der Akkus komfortabel möglich.

3.3.3.7 Komponentenübersicht

Einheit	Baustein	Anzahl pro Roboter	Datenblatt
Mikrocontroller	AT91SAM7S32	1	[Atm05c]
Motoren	SP0618M0104	2	[Nan04a]
Motortreiber	HIP4020	4	[Int05]
Getriebe	Bausatzgetriebe G30	2	[Mik03a]
Energie-Versorgung	Kok 1200	1	[Wes04]
Festspannung	KF33BD	1	[STM04]

Abbildung 3.15: Komponentenübersicht Roboter

Ein Überblick über die wichtigsten Komponenten des Roboters bietet die Tabelle 3.15.

3.3.4 Testboard

Nachdem der Mikrocontroller für die Roboter ausgewählt worden ist (siehe Abschnitt 3.1.3.1), wurde ein Testboard für diesen entwickelt.

Der Mikrocontroller AT91SAM7S32 [Atm05c] benötigt Versorgungsspannungen von 1,8 V und 3,3 V. Da er aber im Gegensatz zum im Feld verwendeten Mikrocontroller von Philips [Phi03] einen internen Spannungsregler für die 1,8 V besitzt, wird extern nur noch ein 3,3 V Spannungsregler benötigt. Auf dem Testboard wird der Spannungsregler KF33BD [STM04] verwendet, der auch später im Roboter Verwendung findet. Somit kann das Testboard mit Spannungen innerhalb des Eingangsbereichs des Spannungsreglers versorgt werden.
Zum Schutz vor zu hohen Strömen ist eine Sicherung vorgesehen, diese bewährte sich bereits bei Tests mit den Schrittmotoren. Außerdem dient eine Diode am Eingang der Spannungsversorgung zum Schutz vor Verpolung.

Weitere Komponenten für den Mikrocontroller sind der Quarz und verschiedene Kondensatoren. Dabei besitzt aber der Mikrocontroller noch die Besonderheit, dass bei der verwendeten Quarzfrequenz von 12 Mhz keine Kondensatoren benötigt werden [Atm05c]. Jedoch werden für den internen Spannungsregler Kondensatoren zur Reduktion von Ripple[10] gebraucht, hier wird im Testboard ein 2,2 μF und parallel ein 1 nF Kondensator eingesetzt. Weiterhin sind mehrere Entstörkondensatoren vorgesehen. Diese sind in der Nähe von ICs zwischen Versorgungsspannung und Masse geschaltet.

Das Testboard enthält einen Taster für den Reset des Mikrocontrollers und einen weiteren Taster, der Eingaben an den Mikrocontroller ermöglicht. Zur Ausgabe stehen vier LEDs zur Verfügung.

Eine serielle Schnittstelle dient der Progammierung und dem Datenaustausch. Der Baustein MAX3232 [Max03] setzt die Pegel des Mikrocontrollers in symmetrische Pegel für die serielle Schnittstelle des PC um. Der Anschluss an den PC ist aufgrund der vorhandenen Sub-D-Buchse über ein serielles Kabel möglich. Der Mikrocontroller AT91SAM7S32 besitzt zwei serielle Schnittstellen. Die eine wird zur Kommunikation mit dem Bootprogramm und dadurch zur Progammierung des Mikrocontrollers benutzt. Die andere dient der Kommunikation mit dem PC. Diese Schnittstellen sind jeweils durch das Umsetzen zweier Jumper wählbar.

Eine weitere Möglichkeit den Mikrocontroller zu programmieren stellt dessen JTAG Schnittstelle dar. Im Schaltplan (im Anhang 6.1) und im Layout (im Anhang 6.8) wurde der Anschluss eines entsprechenden Adapters zwar vorgesehen, aber im Aufbau nicht realisiert, da ein weiterer Hardwareaufwand entstehen würde und die serielle Schnittstelle in Verbindung mit dem Bootprogramm Vorteile bot.

Zum Anbinden von weiteren Testboards oder anderen Komponenten sind alle IO-Pins und auch Schnittstellen des Mikrocontrollers an einen auf dem Testboard enthaltenen Pfostenstecker geführt. Dieser verfügt außerdem über Pins mit Verbindung zu Masse und 3,3 V. Ein weiterer Pin ermöglicht das Zuführen von 1,8 V, um den internen Spannungsregler nicht verwenden zu müssen. Außerdem ist ein Pin direkt mit der Versorgungsspannung des Testboards verbunden, um die Überwachung der Eingangsspannung, die beim Roboter von einem Akku zur Verfügung gestellt wird, zu ermöglichen.

Nach dem Erstellen des Schaltplans (im Anhang 6.1), wurde das Layout (im Anhang 6.8) ohne Verwendung des automatischen Routens mit Eagle erstellt. Das Routen des Layouts von Hand hat sich auch bei anderen gefertigten Platinen bewährt. Dabei wurden Leiterbahnbreiten von 40 mil für die Spannungsversorgung und 10 mil für Signalleitungen verwendet. Außerdem wurde möglichst darauf geachtet, statt 90° Winkeln einheitlich 45° Winkel zu benutzen. Vor dem Ätzen wurden unter Verwendung von Polygonen in Eagle beim Routing ungenutzte Flächen zu Kupferflächen gemacht, wodurch weniger Kupfer weggeätzt werden muss.
Auch das Ätzen fand in Eigenarbeit am Lehrstuhl statt. Zuletzt wurde die bestückte Platine zum besseren Aufstellen auf Abstandsbolzen geschraubt.

[10] Welligkeit

3.3.5 Mechanischer Aufbau

In diesem Abschnitt geht es darum, den Aufbau und die Anordnung der einzelnen Komponenten im Inneren des Roboters zu beschreiben. Die Abbildung 3.16 soll dabei als Veranschaulichung dienen.

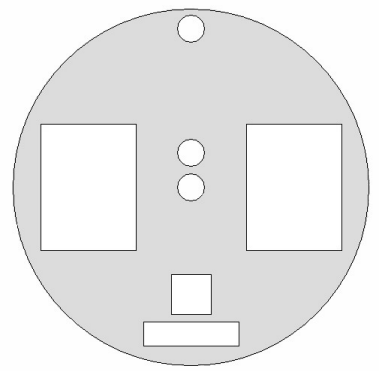


Abbildung 3.16: Grundriss der Bodenplatte mit Aussparungen in Originalgröße

Zu Beginn sei erwähnt, dass die primären Ziele unserer Konstruktion die Unterbringung aller Bausteine und notwendigen Elemente, sowie die platzsparende Fügung dieser sind. Außerdem müssen die Bauteile leichten Erschütterungen, die durch die Bewegung des Roboters entstehen, standhalten können. Es darf im Inneren nichts umkippen oder beschädigt werden. Somit ist gerade die Befestigung der Elemente ein wichtiger Betrachtungspunkt.

Die Grundbausteine

Um die Bewegungsmöglichkeit des Roboters maximal zu gestalten und um dem ästhetischen Gefühl entgegen zu kommen, wird er eine runde Grundfläche erhalten, auf die höchstwahrscheinlich ein zylinderförmiges Gehäuse gesetzt wird.

Die runde Bodenplatte des Prototypen besteht aus Aluminium, ist 2 mm dick und hat einen Durchmesser von 4,5 cm. Um Befestigungsmöglichkeiten für das Gehäuse in den Serienrobotern zu bieten, wird die Bodenplatte dort 4 mm dick werden.

Abbildung 3.17 liefert einen Überblick sämtlicher Maße der Bodenplatte.

Bedingt durch die Maße des Akkus müssen diese aufrecht in der Mitte des Roboters platziert werden. Das bringt auch den Vorteil einer optimalen Gewichtsverlagerung im Inneren mit sich, denn mit rund 20 g ist dieser die schwerste Komponente. Auch die Kommunikationsschnittstellen, Leuchtdiode und Fototransistoren, müssen in der Mitte des Roboters angebracht werden. Da diese in den Boden eingelassen werden müssen, muss Sorge dafür getragen werden, dass sich der Akku oberhalb der mit den Kommunikationselementen verbundenen Bodenplatine befinden kann. Dies wird mit einer Konstruktion

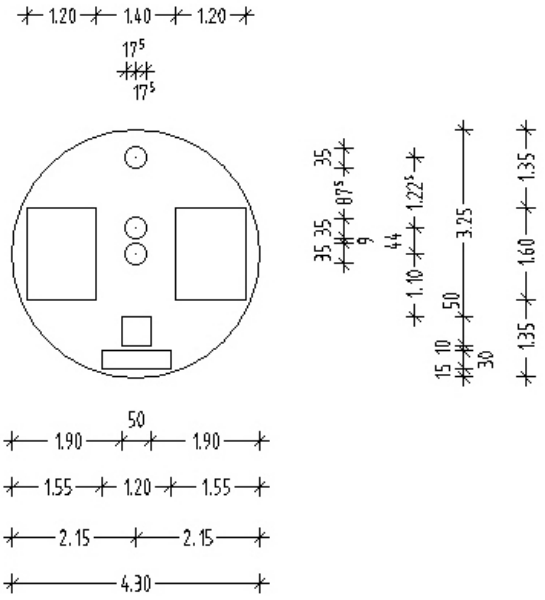


Abbildung 3.17: Die bemaßte Bodenplatte

aus L- und U-Winkeln derart erreicht, dass die Lötungen der Kabel der Bodenplatine die Unterseite des U-Winkels nicht berühren.

Das Material der Winkel ist Kunststoff, da es gerade im Inneren des Roboters wichtig ist, dass keinerlei Kurzschlüsse oder andere elektrische Defekte auftreten können und somit keine leitenden Materialien für diesen Zweck in Frage kommen. Außerdem hat es ein geringes Gewicht und ist einfach zu bearbeiten.
Die gesamte obige Konstruktion bietet eine Befestigungsmöglichkeit für die großen Platinen. Sie werden außen an den Winkeln befestigt, innen klemmt der Akku in dem U-Winkel. Auch die Getriebe können über diese Konstruktion und einen weiteren, kleineren L-Winkel befestigt werden.

Anbringen der Kommunikationsschnittstellen

Um die Kommunikation mit dem Spielfeld gewährleisten zu können, muss sich der erste Fototransistor genau in der Mitte der Bodenplatte befinden, der zweite am vorderen Rand des Roboters (vgl. 3.4.6 und 3.3.6.3). Da diese zur Kommunikation nötigen Elemente auf einer Lochplatine angeordnet sind, ist es einfach sie in die Bodenplatte einzulassen. Hierzu bedarf es bei der richtigen Größe der Löcher (Durchmesser circa 3,5 mm) nicht einmal einer zusätzlichen Befestigung, beispielsweise durch einen Kleber.

Einbau der Stromanschlüsse

Da der Roboter auch über eine Ladefunktion verfügen soll, muss für eine Anschlussbuchse im Inneren Platz bleiben. Dazu wird in die hintere Hälfte der Bodenplatte eine Aussparung mit den Maßen 5 mm x 5 mm gefräst. Direkt dahinter befindet sich ein Schalter, mit dem der Roboter von außen an- und ausgeschaltet werden kann. Zu dessen Unterbringung wird ein 0,3 cm x 1,2 cm großes Loch in die Platte gefräst, durch das dieser gesteckt werden kann. Wenn passgenau gearbeitet wird, muss keine weitere Klebung erfolgen, da der Schalter dann fest in der Aussparung klemmt. Weitere Ausführungen zu diesen beiden Elementen findet man im Abschnitt 3.3.3.6.

Anordnung der Fortbewegungselemente

Kommen wir nun zu der Anordnung der Motoren, Getriebe, Räder und Achsen. Da die beiden Räder getrennt voneinander angesteuert werden sollen, sind zwei Achsen vorhanden (vgl. 3.3.2.1), die in der Mitte nicht verbunden sind. Diese werden von oben in die Bodenplatte in dafür gefräste Aussparungen eingelassen. Zur Führung der Achsen in den Aussparungen dienen Aderendhülsen, die gleichzeitig auch noch den nötigen Spielraum bieten.

Für die Räder und die zugehörigen Getriebe muss an beiden Außenseiten aus der Bodenplatine eine 12 mm x 16 mm große Öffnung gefräst werden.
Beim Prototypen mussten Rad, Felge und Achse jeweils fest miteinander verklebt werden, in der Serienfertigung wird jedoch keine zusätzliche Verbindung notwendig sein, da Felge und Achse aus einem Stück gedreht werden (vgl. 3.3.3.4). Auf der Achse sitzt dann jeweils das Getriebe, auf dem der Motor steckt. Um den Motor, der einen Achsdurchmesser von 1 mm hat, mit der Getriebeschnecke, die einen Innendurchmesser von 0,8 mm hat, verbinden zu können, muss deren Öffnung (vgl. 3.3.3.3) passend aufgebohrt werden.

Damit der Roboter auf den zwei Rädern das Gleichgewicht halten kann, wurde die Bodenplatte auf der Unterseite mit runden Filzgleitern versehen, die geviertelt und je zwei Viertel vorn und hinten zur Stützung aufgeklebt wurden.

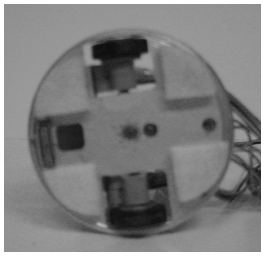


Abbildung 3.18: Die Unterseite der Bodenplatte mit den Filzgleitern

Aufbau

Bei dem Aufbau muss eine gewisse Reihenfolge eingehalten werden, in der die Elemente eingesetzt werden. Zuerst werden die Filzgleiter unter die Bodenplatte geklebt (vgl. Abb. 3.18). Als nächstes müssen die Aderendhülsen in die gefrästen Aussparungen geklebt werden, wobei zu beachten ist, dass die Achse sich vorher in der Hülse befinden muss, da man sie sonst nicht mehr einsetzen kann. Natürlich müssen auch die Räder vorher an der Achse befestigt worden sein.

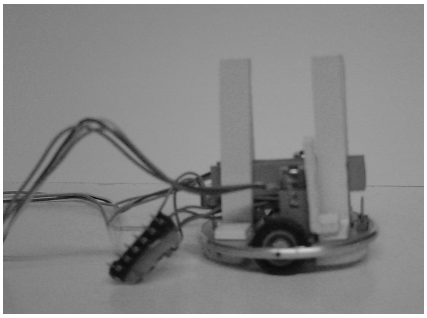


Abbildung 3.19: Bodenplatte mit Getrieben, Rädern, Achsen und Winkeln

Danach wird die Bodenplatine von oben eingesetzt, indem die LED und die Fototransistoren in die vorgesehen Löcher eingelassen werden. Jetzt muss die Winkelkonstruktion aufgebaut werden. Die vier L-Winkel werden an dem U-Winkel befestigt, wobei zwei gegenüberliegende L-Winkel einen kleineren L-Winkel tragen.

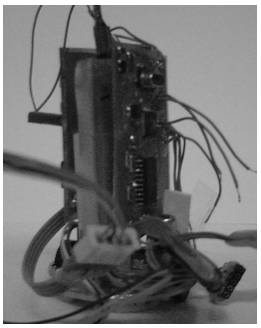


Abbildung 3.20: Der Roboterprototyp (Ansicht hinten)

Nachdem die Buchse und der Schalter in die Bodenplatte eingesetzt sind, kann die Winkelkonstruktion auf diese geklebt werden und die Getriebe an den kleinen L-Winkeln befestigt werden. Jetzt fehlen nur noch die Motoren, der Akku und die Platinen. Der fertige Roboterprototyp ist in den Abbildungen 3.20 - 3.23 zu sehen.

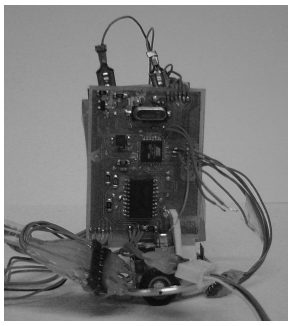


Abbildung 3.21: Der Roboterprototyp (Ansicht links)

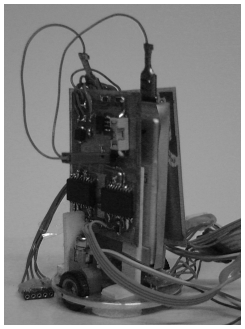


Abbildung 3.22: Der Roboterprototyp (Ansicht vorn)

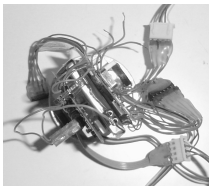


Abbildung 3.23: Der Roboterprototyp (Ansicht oben)

3.3.6 Elektronischer Aufbau

Die Roboter beinhalten eine ganze Reihe elektronischer Komponenten, wie es auch schon bei der Komponentenauswahl in 3.3.3 deutlich wird. Diese gilt es untereinander korrekt zu verbinden und im Roboter zu platzieren.
Aufgrund der Vielzahl der notwendigen Bausteine empfiehlt sich die Nutzung von Platinen. Um die Herstellungskosten gering zu halten, werden diese selbst entworfen und auf der lehrstuhleigenen Ätzanlage hergestellt.

Eine Anforderungsanalyse in 3.3.6.1 zeigt, dass die Komponenten auf mehrere Platinen zu verteilen sind (Abschnitt 3.3.6.2).
Nach gemeinsamem Entwurf wird daher das Routing und die Herstellung einer Bodenplatine in 3.3.6.3, einer Nebenplatine in 3.3.6.4 und einer Hauptplatine in 3.3.6.5 gesondert betrachtet.
Der Entwicklungszyklus wird wie gewöhnlich durch Tests, beschrieben in Abschnitt 3.3.6.7, abgeschlossen.

3.3.6.1 Anforderungen

Die Anforderungen an die Platinen werden durch mechanische und elektronische Entscheidungen und Notwendigkeiten geprägt.
Wie im Roboterlayout in 3.3.5 gemäß den Anforderungen an den Roboter aus 3.3.1 ersichtlich, besteht eine einschneidende Beschränkung bezüglich der Dimensionierungsmöglichkeiten der Platinen. Nutzflächenmaximierend ist eine zum Akku parallele Platinenplatzierung, die eine Größe von 38 mm x 50 mm bei kreisbogenförmiger Tiefe erlaubt.

Weiterhin sind eine Reihe von Komponenten unterzubringen, unter anderem der Mikrocontroller AT91SAM7S32, vier Motortreiber, die Festspannungsversorgung sowie die Kommunikationskomponenten. Letztere müssen so untergebracht sein, dass die Schnittstellen zur optischen Kommunikation an den dazu spezifizierten Stellen unter dem Roboter zur Verfügung stehen.
Ähnliches gilt für die weiteren Schnittstellen, wie die zu den Motoren, dem Akku, dem Ein-/Ausschalter und dem Ladestecker.

Diesen zum Teil divergierenden Anforderungen gerecht werdend ist ein Kompromiss zu erarbeiten.

3.3.6.2 Aufteilung der Komponenten

Obwohl jede der verwendeten Komponenten in kleiner Bauform erhältlich ist, ist eine Platzierung selbst auf einer zweiseitigen Platine dieser Größe am Lehrstuhl nicht möglich. Daher wird eine Lösung mit mehreren Platinen zwangsläufig notwendig.

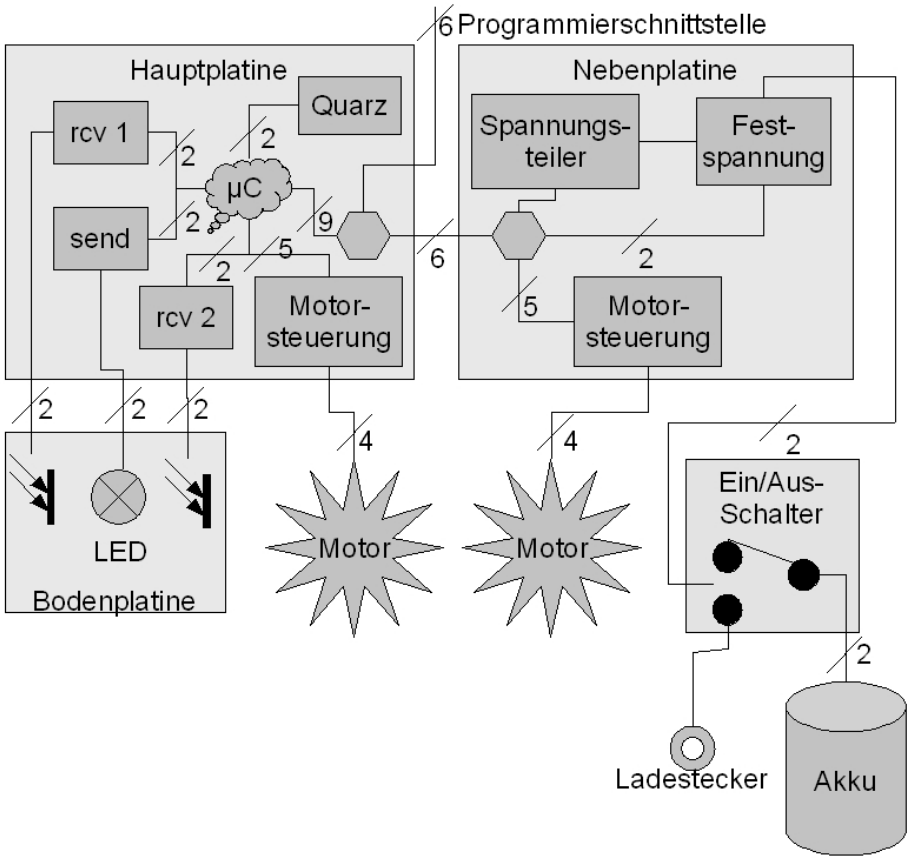


Abbildung 3.24: Platinenaufteilung

Durch günstige Aufteilung der Bauteile ist eine Erfüllung aller gestellten Anforderungen möglich. Dazu werden drei Platinen verwendet:
Eine Bodenplatine (Abschnitt 3.3.6.3) umfasst die optischen Kommunikationsschnittstellen. Eine Nebenplatine (Abschnitt 3.3.6.4) beherbergt neben der Steuerung des zweiten Motors die Spannungsversorgung. Eine Hauptplatine schließlich (Abschnitt 3.3.6.5) umfasst die Mikrocontrollerschaltung und die Steuerung für den zweiten Motor.

Die Haupt- und Nebenplatine werden respektive links und rechts vom Akku platziert, ihre Gesamtgröße kann dadurch maximiert werden.

Die Aufteilung orientiert sich dabei an folgenden Faktoren:

- die Positionierung der Schnittstellen: Motorschnittstellen sind beispielsweise auf Platinen nahe der anzusteuernden Motoren untergebracht, um Anschlusskabel kurz zu halten.
- die möglichst gleichmäßige Verteilung der Komponenten: Durch eine gleichmäßige Verteilung soll das Routing der Platinen erleichtert werden.

- die Minimierung der Verdrahtung zwischen den Platinen: Verdrahtungen sind fehleranfällig und unhandlich in der Verarbeitung.

3.3.6.3 Bodenplatine

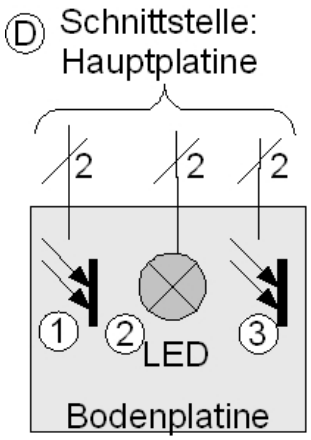


Abbildung 3.25: Bodenplatine

Da die Kommunikationselemente direkt in die Bodenplatte eingelassen sind, bietet es sich an, die Bodenplatine direkt dadrüber anzubringen. Aufgrund der damit verbundenen geringen Größe der Platine (28 mm x 6 mm) können keine weiteren Komponenten auf ihr untergebracht werden.

Da nur drei ausschließlich bedrahtete Komponenten verwendet werden, bietet sich die Verwendung einer Lochrasterplatine an. Eine Ansteuerung der optischen Komponenten erfolgt durch die Mikrocontrollersteuerung der Hauptplatine.

3.3.6.4 Nebenplatine

Gemäß der Aufteilung der Komponenten zu den funktionalen Einheiten Festspannungsregler, Spannungsteiler und Motorsteuerung (Abb. 3.26) werden die einzelnen Bauteilgruppen im Folgenden erläutert.

Festspannungsregler

Eine Unterbringung des Festspannungsreglers auf der Nebenplatine bietet sich aus mehreren Gründen an:

- Mikrocontroller und Festspannungsregler sind beide flächenintensiv und daher auf die Platinen aufzuteilen.

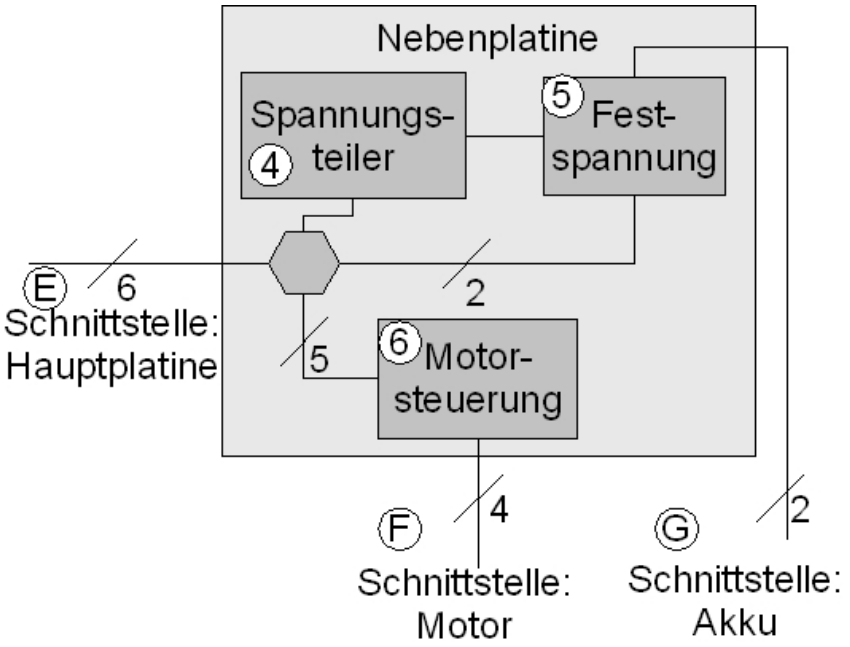


Abbildung 3.26: Nebenplatine

- Es sind keine zusätzlichen Verbindungsleitungen zwischen den Platinen notwendig, da die Stromversorgung auf beiden benötigt wird.
- Die im Vergleich zur Versorgungsspannung höhere Akkuspannung wird nicht in der Nähe vom Mikrocontroller eingesetzt.

Die Festspannungsversorgung umfasst neben der Kernkomponente KF33DB zwei Kondensatoren, einen vorgeschalteten 100 nF Kondensator, der mögliche Schwankungen seitens der Spannungsquelle, und einen nachgeschalteten 2,2 μF Elko-Kondensator, der Bedarfspitzen seitens der Spannungsabnehmer glättet.

Darüber hinaus wird die empfindliche Elektronik des Roboters durch eine 1 A Sicherung geschützt.

Spannungsteiler

Da die Akkuspannung lediglich direkt auf der Nebenplatine verfügbar ist, ist ein Spannungsteiler ebenfalls hier zu platzieren. Dieser ermöglicht das Messen der Akkuspannung mit dem Mikrocontroller, dessen Messbereich durch die Referenzspannung des Festspannungsreglers begrenzt ist.

Er besteht lediglich aus zwei hochohmigen Widerständen, wodurch der Stromverbrauch minimiert wird. Beide sind mit 1 MΩ gleich groß gewählt, damit zwischen den Widerständen gegenüber Masse genau die halbe Akkuspannung gemessen werden kann.

Diese halbierte Akkuspannung ist zu einem Eingang des A/D-Wandlers des Mikrocontroller zu führen.

Motorsteuerung

Schon in 3.3.6.2 wird deutlich, dass es günstig ist, eine Motortreibereinheit auf der Nebenplatine zu platzieren. Diese besteht aus zwei HIP4020 Bausteinen.

Da diese jedoch am Ausgang ungefähr 3 V liefern, die Motoren hingegen nur 1,96 V benötigen, ist eine Spannungsreduktion per Widerstand, wie in 3.3.3.2 schon erläutert, notwendig.

Der Motor braucht also 1,96 V bei 140 mA und hat dementsprechend einen Widerstand von:

$$R_{Motor} = \frac{U_{Motor}}{I_{Motor}} = \frac{1,96V}{140mA} = 14\Omega$$

Nach der zweiten Kirchhoffschen Regel (Maschenregel) muss sich die Spannung proportional zum Widerstandswert auf alle in Reihe geschalteten Widerstände verteilen, in Summe der Gesamtspannung (U_{ges}) entsprechend. Dabei ist der Stromfluss durch alle Widerstände gleich groß.

So muss für einen vorgeschalteten Widerstand gelten, dass durch ihn ein Strom von $I_R = I_{Motor} = 140mA$ fließt und $U_R = U_{ges} - U_{Motor} \approx 1V$ Spannung abfällt. Für den Widerstand resultiert nun:

$$R = \frac{U_R}{I_R} = \frac{1V}{140mA} = 7,1\ldots\Omega \approx 10\Omega$$

Für jede Spule wird dementsprechend ein 10 Ω Widerstand zwischengeschaltet.

Die Motorsteuerung bildet eine Funktionseinheit und wird stets als Ganzes ein- oder ausgeschaltet. So werden die jeweils zwei HIP-Enable Eingänge (ENA, ENB) der Motortreiber jeweils zusammengeschaltet.

Ein HIP4020 steuert jeweils eine Motorspule an. Da die Eingänge eines HIPs invers zueinander sind, können dessen Steuereingänge (A1, $\overline{\text{B1}}$) in unserer Anwendung ebenfalls zusammengeschaltet werden.

Insgesamt sind so durch den Mikrocontroller zwei Steuerleitungen und eine Enable-Leitung zu steuern.

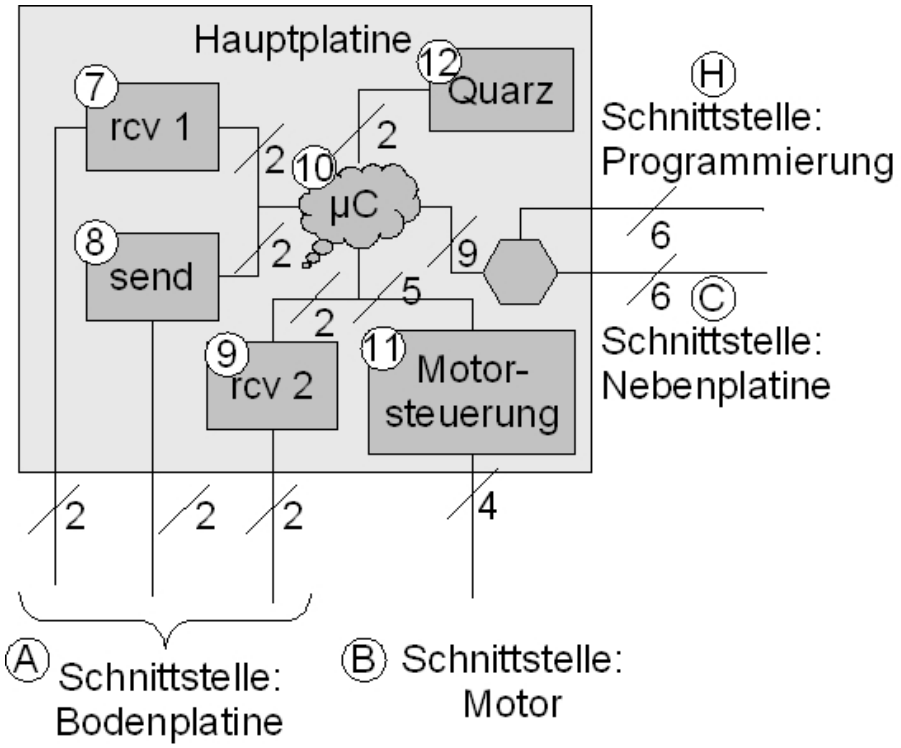


Abbildung 3.27: Hauptplatine

3.3.6.5 Hauptplatine

Das Herzstück der Hauptplatine ist der Mikrocontroller. Trotz seiner geringen Größe braucht er mit seinen diversen zu verbindenden Ausgängen recht viel Platz auf der Platine und stellt alleine schon beim Routing eine eigene Herausforderung dar. Letzteres gilt insbesondere auch für das Löten und Testen.

Spannung und Masse werden mehrfach in den Mikrocontroller geführt, zusätzlich ist eine intern erzeugte Arbeitsspannung über mehrere Pins an verschiedenen Seiten wieder zuzuführen.
Da jedoch an allen Seiten des Controllers weitere Leitungen liegen, gilt es die nun nötigen Überschneidungen geschickt zu legen, um größtmöglichen Nutzen bei geringem Aufwand zu erhalten. Verschärft wird diese Anforderung durch die festgelegten Pins der notwendigen Peripherie des Mikrocontrollers. Eine optimale Pinbelegung für eigene Peripherie in der Programmierung konnte weitere unnötige Überschneidungen und die damit verbundenen Durchkontaktierungen vermeiden.

In den folgenden Abschnitten wird, ähnlich zur Nebenplatine in 3.3.6.4, auf die Komponenten dieser Platine und ihre Positionierung eingegangen, sowie dazugehörige Besonderheiten erläutert.

Vor der Platzierung der Bausteine wurden zwei grundlegende Entscheidungen getroffen. Wegen der kreisbogenförmigen Tiefe der begrenzenden Roboterwand sind die großen Bausteine möglichst mittig zu platzieren.
Zusätzlich soll die Rückseite der Platine möglichst nur zum Führen von Leiterbahnen dienen, damit die Platinenhalterungen dort freier und ohne Erhöhung der Bautiefe positioniert werden können.
Diese zwei Entscheidungen haben entsprechende Konsequenzen für die im Folgenden behandelten Punkte.

Mikrocontroller

Da der Mikrocontroller mit allen anderen Komponenten verbunden wird (vgl. Abbildung 3.27 Nr.10), hat er auf der Platine eine sehr zentrale Lage.

Die eben angesprochenen Versorgungsleitungen werden teilweise unter den Controller gelegt, um mehr Platz für die steuernden Ausgangsleitungen zur Verfügung zu haben.
Zusätzlich werden direkt neben dem Controller an den Pins PGMEN1-3 und TST kleine Pads herausgeführt, da diese Pins für einen vollständigen Reset des Controllers (inklusive des Flashs) nötig sind.

Mikrocontroller Peripherie

Einige weitere Bausteine sind notwendig, um den Mikrocontroller betreiben zu können. Angaben zu diesen lassen sich dem Datenblatt [Atm05c] entnehmen. Dazu gehören ein Quarz und zwei Kondensatoren (siehe 3.3.4).
Diese Komponenten müssen möglichst nah am Controller platziert werden.
Bei dem Quarz ist zusätzlich noch auf möglichst gleich lange Leiterbahnen zu achten, damit die Schwingung nicht verfälscht wird. Ebenso dürfen keine weiteren Bahnen zu nah am Quarz sein, da hier potentiell die Gefahr des Übersprechens entsteht.

Anordnung der Schnittstellen

Die Platzierung der Schnittstellen ist hauptsächlich durch den mechanischen Aufbau bestimmt. Die Verbindungskabel sollen möglichst kurz sein und direkt die einzelnen Komponenten verbinden. Es ergibt sich folgende Anordnung:

Schnittstelle zum Motor: Die Motorsteuerung wurde auf beide Platinen verteilt, um möglichst nah an den Motoren zu sein (vgl. 3.3.6.2). Diesem Gedanken weiter folgend befindet sich die Schnittstelle am unteren Ende der Platine.

Schnittstelle zur Bodenplatine: Es bietet sich an, diese Schnittstelle aus demselben Grund ebenfalls ans untere Ende zu setzen.

Schnittstelle zur Nebenplatine: Die Breite des Roboters stimmt mit der des Akkus überein. Für die Verbindung zur Nebenplatine können die Kabel also nur oben über den Akku geführt werden.

Programmierschnittstelle: Die Programmieranschlüsse des Mikrocontrollers liegen auf der rechten Seite. Da es keine weitere Einschränkungen gibt, kann die Schnittstelle dort beliebig positioniert werden.

Diese Aufteilung der Schnittstellen legt es nahe, die gesamten Kommunikationskomponenten (und damit auch die Bodenschnittstelle) linksseitig und die Schnittstelle zur Nebenplatine rechtsseitig zu platzieren.

Positionierung der Motortreiber

Die Beschaltung und Ansteuerung der Motortreiber HIP4020 wurde grob bereits in erläutert.

Problematisch auf der Hauptplatine ist jedoch der Platzbedarf dieser Bausteine. Zudem ähneln sich die Bausteine in der Verdrahtung sehr, jedoch auf eine Art, die viele Durchkontaktierungen nötig macht.

Da der mittlere Teil der Platine nicht durch die spätere Befestigung in Anspruch genommen wird, kann für diesen auf die Forderung nach nur einseitiger Bestückung mit Bauteilen verzichtet werden. Folglich kann jeweils ein HIP4020 auf jeder Seite platziert werden.

Dies hat eine Reihe von Vorteilen. Da Durchkontaktierungen, wie die Nebenplatine zeigt, notwendig sind, können diese nun gleich auch zur Ansteuerung auf der Unterseite verwendet werden. Weiterhin ergibt sich eine günstige Lage der Pins, die kürzere Leitungen und weniger Durchkontaktierungen erlaubt. Beide Bausteine können horizontal zentriert zwischen Controller und Schnittstelle platziert werden.
Als Folge hat sich nicht nur der Platzbedarf (inklusive Leitungen) minimiert, sondern auch die Anzahl der Durchkontaktierungen ist geringer als auf der Nebenplatine.

Anordnung der Kommunikation

Dank der flachen Bauweise der dafür notwendigen Komponenten kann die Kommunikation die Unterseite der Platine mitnutzen.

Während die LED-Schaltung nur aus dem dafür vorgesehenen Widerstand besteht, sind die Transistorschaltungen mit mehreren Widerständen ausgestattet. Eine Ergänzung mit 0 Ω Widerständen vereinfacht hinterher eine Anpassung des Steuergesamtwiderstandes, da eine Reihe von Widerstandswerten nur durch zwei in Reihe geschalteten Widerständen realisiert werden kann. die 0 Ω Widerständen können, falls eine Änderung später nötig sein sollte, durch einen anderen Wiederstand ersetzte werden.

3.3.6.6 Herstellen der Platinen

Die Besonderheiten der Platinen stellen einige Herausforderungen an die Produktion. Die in 3.1.2 angesprochene Möglichkeit, extern hergestellte Folien zu nutzen, ist hier wegen der sehr dünnen Leiterbahnen dringend zu empfehlen, da ein ordentliches Ätzen in dieser Größenordnung mit den vom vorhandenen Drucker hergestellten Vorlagen fast unmöglich ist.

Auch das Bestücken stellt sich kompliziert dar. Die Platine wird vor der Bestückung zurecht gesägt, ein Rand ist dabei nicht vorgesehen. Das Fixieren der Platine bei dieser geringen Größe ist dadurch schwierig. Übliche Halterungen treten in Konflikt mit den nahe am Rand platzierten Bauteilen. Zusätzlich sind die zwei Halterungen dann so nah beieinander, das sie selbst das Arbeiten zum Teil behindern. Diese Problematik kommt besonders während der Bestückung der Rückseite zum Tragen, die zusätzlich zum fehlenden Fixieren eine Schräglage der Platine verursacht.

3.3.6.7 Test der Platinen

Eine stetige Überprüfung der Funktionsfähigkeit der Platinen stellt sicher, dass Fehler früh erkannt werden, und dass die Anzahl möglicher Fehlerquellen überschaubar bleibt. Insbesondere bei Problemen während der Herstellung der Platinen müssen diese besonders sorgfältig geprüft werden.

Die Platinen durchlaufen dabei folgende Überprüfungen, ggf. mit mehreren Iterationen:

1. Durchmessen der Leiterbahnen:

   Unmittelbar nach dem Ätzen werden alle Leiterbahnen mittels Ohmmeter kontrolliert, ob sie von Anfang bis Ende Kontakt haben. Darüber hinaus werden alle unmittelbar benachbarten Flächen überprüft, ob sie elektrisch von der betrachteten Leiterbahn getrennt sind.

2. Durchmessen nach Durchkontaktierung:

   Nach Anbringen der Durchkontaktierungen werden diese wie beim ersten Punkt geprüft.

3. Durchmessen nach Bestückung:

   Wenn die Bauteile auf der Platine angebracht sind, gilt es, die Kontakte nochmals zu prüfen. Dabei sind die Datenblätter der Bauteile heranzuziehen, um die Innenwiderstandswerte der Bauteile in Erfahrung zu bringen. So können nun auch deren Installationen geprüft werden: Vom Widerstand, dessen Wert direkt erkennbar ist, bis hin zum Spannungsstabilisator, der auf der Nebenplatine beispielsweise mit einen Innenwiderstand von etwa 34 kΩ gemessen wurde.

4. Durchmessen nach Verkabelung / Einbau:

Insbesondere der Aufbau im Roboter lässt bei der Endmontage Probleme entstehen. So floss zum Beispiel durch einen über das anliegende Motorgehäuse überbrückten Widerstand ein zu hoher Strom am Motor. Weiterhin erschwerten Leitungen rund um Kabelanlötstellen deren kurzschlussfreie Montage. Verbesserungen im Layout waren somit nötig.

Darüber hinaus sind aufeinander aufbauende Einsatztests nötig, um auch hier gegebenenfalls Fehler und Probleme aufzudecken:

1. Test der Stromversorgung:

   Der Spannungsstabilisator wird über eine externe Spannungsquelle probehalber mit verschiedenen Spannungen versorgt. Die ausgehende Spannung kann mit einem Voltmeter geprüft werden.
   Ein derart umsichtiges Vorgehen schützt empfindliche Komponenten, falls eine fehlerhafte Stromversorgung vorliegt, denn sie wurden bei diesem Test von der Spannungsversorgung getrennt, um keiner eventuell auftretenden Überspannung ausgesetzt zu sein.

2. Test der Motorsteuerung:

   Über eine externe Testschaltung ist insbesondere die Motorsteuerung der Nebenplatine vollständig zu testen, da alle Ansteuerungen nach Außen geführt sind. Dieser Test stellt folgendes sicher:

   - die Motoren werden nicht mit zu hoher Spannung betrieben;
   - die Mikrocontrollerausgänge werden durch die Motoransteuerung nicht zu stark belastet.

   Als weiterer Schritt der Motoransteuerungstests werden dann die Motoren direkt angeschlosen, um die tatsächliche Betriebslast zu erreichen.

3. Test des Mikrocontrollers:

   Der Mikrocontroller kann in Betrieb genommen werden. Am Quarz wird gemessen, dass die Inbetriebnahme erfolgreich ist, da dieser mit der Zielfrequenz schwingen muss, im Testaufbau also mit etwa 12 MHz.

4. Integrativer Test:

   Die Komponenten werden angeschlossen, über den Mikrocontroller sind dann Motoren und LED anzusteuern sowie Fototransistoren auszulesen.

Der Abschluss ist also ein alles umfassender Test; haben die Platinen ihn erfolgreich überstanden, so ist ihre Einsatzfähigkeit sichergestellt.

3.3.7 Programmierung

Grundlegende Designentscheidung der Projektgruppe R_3D_3 war, dass die Spielintelligenz in den Spielfeldern zu realisieren ist, da diese

1. über beliebig viel Energie verfügen und so
   a) leistungsfähigere Prozessoren versorgen können, und
   b) diese beliebig lange betreiben können, ohne dass es in Folge von Energieversorgungsengpässen zu Datenverlusten kommt;
2. über schnelle Kommunikationsverbindungen zueinander verfügen und daher besser komplexere Nachrichten untereinander austauschen können.

Entsprechend diesem Designgedanken implementiert der Roboter kaum Spielintelligenz. Lediglich das Kommunikationsprotokoll und die Fehlerkorrektur gestalten sich anspruchsvoller.
Die Mikrocontrollerprogrammierung erfolgt hardwarenah, daher wird die Programmierung an dieser Stelle im Hardwareabschnitt vorgestellt.

Ausgehend von einem Top-Down Ansatz - vorgestellt in 3.3.7.1 um einen Überblick über Ablauf und Funktionsumfang der Programmierung zu gewähren - werden in den folgenden Abschnitten einige Funktionen und Aspekte der Programmierung genauer vorgestellt:

- die Akkuüberprüfung in 3.3.7.2,
- der Sleepzustand in 3.3.7.3,
- die Kommunikationsprotokollumsetzung in 3.3.7.4,
- die Nachrichtenauswertung in 3.3.7.5,
- die Motoransteuerung in 3.3.7.6,
- der Fehlerkorrekturalgorithmus in 3.3.7.7, sowie schließlich
- die Softwarevalidierung in 3.3.7.8.

Die Programmierung erfolgt in C, Sequenzen des Quellcodes werden präsentiert.

3.3.7.1 Robosoft

Die Roboter iterieren über die folgenden Arbeitsschritte:
Zu Beginn steht ein Roboter auf einem Spielfeld und wartet auf einen Befehl. Sobald dieser eintrifft, d.h. sobald das Spielfeld einen Befehl sendet, wird er bestätigt und ausgeführt. Nach Ausführung beginnt ein erneutes Warten.

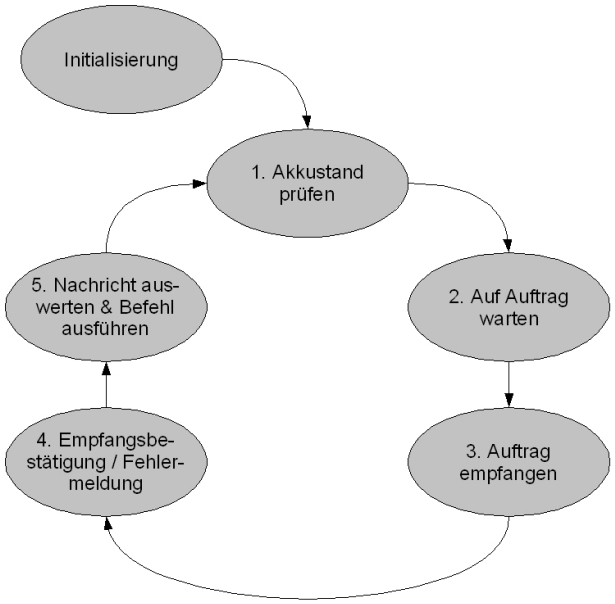


Abbildung 3.28: Programmablauf Roboter

Auf dieser Basis lässt sich die Struktur der Software - wie Abb. 3.28 zeigt - in fünf Teilschritte gliedern, denen nur eine einmalige Initialisierungsphase vorangeht.

In den einzelnen Teilschritten werden dabei folgende Arbeiten ausgeführt:

Initialisierung:

Um auf Komponenten wie die LED und den ADC (Analog/Digital Converter)[11] zurückzugreifen, müssen zuvor einige Registerwerte des Prozessors gesetzt werden[12]. Darüber hinaus ist es wichtig, die Robotermotoren bzw. deren Motortreiber in einen definierten Anfangszustand zu setzten (vgl. 3.3.3.2). Beide Aufgaben werden in der Initialisierung gemäß Abbildung 3.29 ausgeführt.

```
10  // *** Roboterinitialisierung ***
    hip_init();
12  led_init();
    adc_init();
```

Abbildung 3.29: Datei: main.c Initialisierung

[11] Analog/Digital Converter bilden analoge Spannungen zwischen Ground und einer Referenzspannung auf Binärzahlen ab.

[12] Alle Mikrocontrollerkomponenten werden über spezielle Register angesteuert und kontrolliert, näheres ist dem Datenblatt zu entnehmen [Atm05c].

Akkustand prüfen (1):

Damit der Roboter korrekt arbeitet, wird vor jedem Arbeitszyklus gemäß Abb. 3.30 einmal geprüft, ob genügend Energie zur Verfügung steht. Ist die Spannung der Batterie kritisch, so erkennt der Roboter dies. Er übermittelt dem Spielfeld eine entsprechende Fehlermitteilung, da ein fehlerfreies Arbeiten nicht mehr lange sichergestellt ist.

```
18      // *** 1. Akkustand pruefen ***
        if (!robo_check_battery()) {
20        robo_msg_send(MESSAGE_EMPTY_BATTERY);
          break;
22      }
```

Abbildung 3.30: Datei: main.c Akkustand prüfen

Auf Auftrag warten (2) und Auftrag empfangen (3):

Der Roboter wird in einen Sleepzustand versetzt (Abb. 3.31), um möglichst stromsparend auf eine vom Spielfeld initiierte Kommunikation zu warten. Dabei können nahezu alle Komponenten des Roboters inklusive vieler Mikrocontrollerkomponenten ausgeschaltet werden. Lediglich eine Empfangseinheit zur Kommunikation wird betrieben.

Durch die Initiierung einer Kommunikation angeregt (vgl. 3.2.3) wird dieser energiesparende Zustand verlassen und mittels der folgenden Kommunikation eine Nachricht (msg) empfangen.

```
24      // *** 2. Auf Auftrag warten ***
        robo_sleep();
26
        // *** 3. Auftrag empfangen ***
28      int msg;
        robo_msg_rcv(&msg);
```

Abbildung 3.31: Datei: main.c Auftrag

Empfangsbestätigung / Fehlermeldung (4):

Gemäß des Kommunikationsprotokolls in 3.2.3 wird der Empfang einer Nachricht mit einer IDENTIFY Message bei erfolgreicher Übertragung (Zeile 36) und einer COMM_ERROR Message bei einem detektierten Kommunikationsfehler (Zeile 33) bestätigt (Abb. 3.32).

```
    // *** 4. Empfangsbestaetigung / Fehlermeldung ***
32  if ((msg & MESSAGE_TYPE_MASK) == MESSAGE_RES_ID){
      robo_msg_send(MESSAGE_COMM_ERROR);
34  }
    else {
36    robo_msg_send(MESSAGE_IDENTIFY);
    }
```

Abbildung 3.32: Datei: main.c Nachricht bestätigen

Nachricht auswerten und Befehl ausführen (5):

Zuletzt ist die in der Nachricht codierte Anweisung auszuführen (Abb. 3.33).

Einige wichtige Programmaspekte werden nun näher betrachtet.

3.3.7.2 Akkuüberprüfung

Die Überprüfung des Akkus hat zum Ziel, dass der menschliche Spieler rechtzeitig über eine schwache Stromversorgung im Roboter informiert werden kann.
Dies hat durch den Host-PC zu erfolgen. Dazu muss dieser jedoch - über die Spielfelder - von der schwachen Akkuspannung in Kenntnis gesetzt werden. Daher wird die Spannung regelmäßig geprüft und das Spielfeld ggf. über einen Mangel derselbigen unterrichtet (Abb. 3.30), damit dieses die Nachricht weiterleiten kann.

Idee

Die Vorgehensweise zur Prüfung der Akkuspannung ist simpel (Abb. 3.34).

Die halbe Akkuspannung, die durch den Spannungsteiler (vgl. 3.3.6.4) zur Verfügung steht, wird gemessen und mit einem Schwellwert (engl. THRESHOLD) verglichen (Abb. 3.34 Zeile 2).
Fällt sie unter $1,8\,V$ - und somit respektive die Akkuspannung unter $3,6\,V$ - so kann ein korrekter Betrieb nicht länger garantiert werden[13]. Daher wird eine Fehlermeldung gesendet (Abb. 3.30 Zeile 20).
Das Programm wird dadurch jedoch nicht abgebrochen, ggf. wird nur bei der nächsten Durchführung die Fehlermeldung wiederholt, bis der Roboter aus dem Spiel genommen wird.

[13] Der Festspannungsregler benötigt für eine sichere Spannungsversorgung $3,6\,V$ Eingangsspannung.

```
    // *** 5. Auswerten & Befehl ausfuehren ***
40  switch (msg & MESSAGE_TYPE_MASK) {
      case MESSAGE_IDENTIFY: // ID angefordert
42      break; // als Bestaetigung geschehen
      case MESSAGE_ROTATION: // Rotationsnachricht
44      switch (msg & MESSAGE_DATA_MASK) {
          case MESSAGE_ROTATION_LEFT:
46          robo_left(1);
            break;
48        case MESSAGE_ROTATION_RIGHT:
            robo_right(1);
50          break;
          case MESSAGE_ROTATION_UTURN:
52          robo_left(2);
            break;
54        default:
            robo_msg_send(MESSAGE_COMM_ERROR);
56          break;
        }
58      break;
      case MESSAGE_MOVEMENT: {// Bewegungsnachricht
60      int steps = (msg & MESSAGE_STEPS_MASK) >> 1;
        switch (msg & MESSAGE_DIRECTION_MASK) {
62        case MESSAGE_MOVEMENT_FORWARD:
            robo_forward(steps);
64          break;
          case MESSAGE_MOVEMENT_BACKWARD:
66          robo_backward(steps);
            break;
68        default:
            robo_msg_send(MESSAGE_COMM_ERROR);
70          break;
        }
72      break;
      }
74    default: //Komm-Fehler
        robo_msg_send(MESSAGE_COMM_ERROR);
76      break;
    }
```

Abbildung 3.33: Datei: main.c Nachricht auswerten

```
int robo_check_battery() {
  if (adc_get_vdd() < THRESHOLD_BATTERY){
    return 0;
  }
  return 1;
}
```

Abbildung 3.34: Datei: robo.c Akkuüberprüfung

Messen der Akkuspannung

Um die halbe Akkuspannung zu messen, kann einer der A/D-Wandler Eingänge des Mikrocontroller genutzt werden. Dieser bildet die Eingangsspannung - hier also die halbe Akkuspannung - im Vergleich zu einer Referenzspannung auf den Wertebereich zwischen 0 und 1023 ab (10 Bit Auflösung).

Da die Referenzspannung bekannt, stabil und höher als die zu messende Spannung sein muss, wird dazu die Versorgungsspannung nach dem Festspannungsregler verwendet. Diese kann zwar bei zu schwacher Batterie einbrechen und somit die Ergebnisse verfälschen, jedoch geschieht dies erst nach einigen Fehlermeldungen (der THRESHOLD-Wert ist entsprechend hoch zu wählen).

Die Messung erfolgt dabei in fünf Schritten:

1. der ADC (ADC = Analog/Digital Converter) des Mikrocontrollers wird eingeschaltet (standardmäßig aus, um Energie zu sparen);
2. die Messung wird gestartet;
3. auf das Messergebnis wird gewartet;
4. das Messergebnis wird ausgelesen;
5. der ADC wird wieder ausgeschaltet.

Die Schritte erfolgen im Wesentlichen durch das Setzen von Registern, in denen bestimmte Bits Aktionen anstoßen. So wird z.B. das Bit 1 des Channel Enable Registers (AT91C_ADC_CHER) auf „true“ gesetzt, um den ersten Kanal des ADC einzuschalten.

Um auf die Auswertung des ADC in Schritt 3 zu warten, kann - da keine weiteren Aufgaben in dieser Zeit durch den Roboter zu erledigen sind - auf ein sog. „busy waiting“ zurückgegriffen werden. Durch die Anweisung

```
while ((*AT91C_ADC_SR & AT91C_ADC_EOC1) != AT91C_ADC_EOC1);
```

wird das Statusregister des ADCs (AT91C_ADC_SR) überwacht. Durch die Und-Verknüpfung wird dabei nur auf das Bit des ersten Kanals geachtet. Den Abschluss der

Umwandlung signalisiert der Mikrocontroller durch setzen des End of Conversion Bit des ersten Kanals (AT91C_ADC_EOC1).

Sobald also die Konvertierung des analogen Signals abgeschlossen ist, wird das Programm fortgesetzt. Das entsprechende Statusbit wird dabei durchs Lesen des Messergebnisses indirekt zurückgesetzt und kann so das Ende einer weiteren Konvertierung anzeigen.

Auf diese Weise ist nun die Akkuspannung dem Roboter bekannt. Er kann gemäß obiger Ausführung handeln und ggf. den menschlichen Spieler informieren lassen.

3.3.7.3 Sleepzustand

Zum aktuellen Zeitpunkt wird der Sleepzustand noch durch ein busy waiting simuliert. Ziel aber ist es, weite Bereiche des Roboters incl. seines Mikrocontrollers abzuschalten, um so Energie zu sparen. Der Roboter soll dann durch einen vom Spielfeld initialisierten Interrupt über den zentralen Fototransistor aufgeweckt werden.

3.3.7.4 Kommunikation

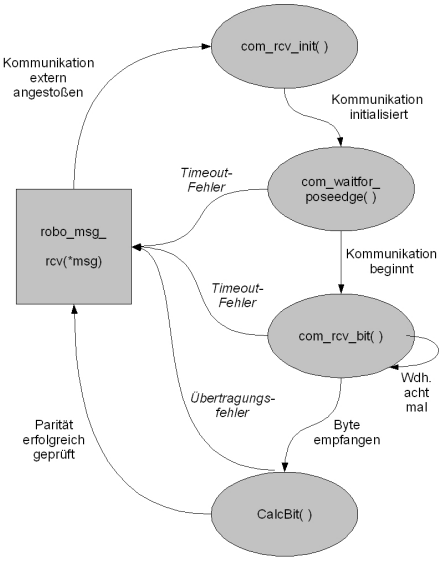


Abbildung 3.35: Nachrichtenempfang

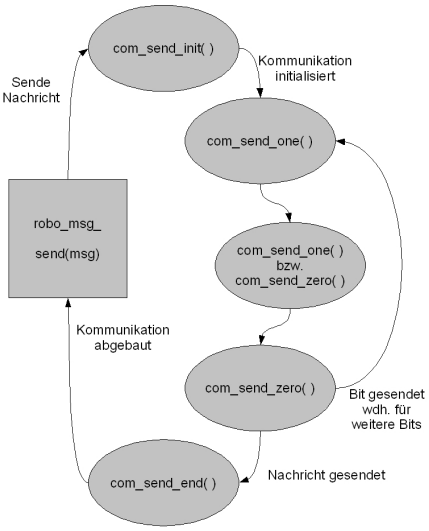


Abbildung 3.36: Nachrichtenversand

Das Kommunikationsprotokoll wird in zwei Schichten realisiert, zum Einen eine Schicht hardwarenaher Prozeduren, die das Senden und Empfangen von Einsen und Nullen sowie den Verbindungsaufbau und -abbau übernehmen (Ellipsen in den Abbildungen 3.35 und 3.36), zum Anderen eine Schicht von Steuerroutinen, die die Kommunikation kontrollieren (Rechtecke und Pfeile in derselben Abbildungen).

Das Protokoll selbst wird in 3.2.3 vorgestellt, an dieser Stelle werden nur Besonderheiten der Programmierung zur Umsetzung aufgezeigt.

Der Timer

```
1  void com_wait(int times){
     int i;
3    for(i = 0; i < times; i++){
       while( (*AT91C_PITC_PISR & 0x1) != 0x1);
5      *AT91C_PITC_PIVR;
     }
7  }
```

Abbildung 3.37: Datei: com.c Realisierung der Zielfrequenz

Eine Funktion namens com_wait() stellt für die Kommunikation ein Taktsignal bereit.

Um mindestens $8\,kHz$ Abtastfrequenz und etwa $4\,kHz$ Sendefrequenz zu erreichen, wird ein Timer (PIT = Periodic Interval Timer) genutzt, der ein $12kHz$ Signal erzeugt. Dieses wird direkt genutzt, um die Abtastung zu betreiben. Für das Senden wird dieses Signal um Faktor 3 reduziert.

Da der Systemtakt mit $12\,MHz$ und der PIT-Zähler mit $\frac{1}{16}$ dieser Geschwindigkeit läuft, muss ein Zähler mit dieser Frequenz bis $COUNTER$ zählen, bevor ein entsprechendes Signal gegeben wird, wobei für $COUNTER$ gilt:

$$\frac{\frac{1}{16} * 12\,MHz}{COUNTER} = 12\,kHz$$

Ergo muss der PIT bis $COUNTER \approx 63$ zählen.

Dieser Wert wird dem PIT bei der Initialisierung mitgeteilt. Von nun an zählt er automatisch bis zu diesem. Zeile 4 von Abbildung 3.37 zeigt, wie per busy waiting auf das nächste Taktsignal gewartet wird, ggf. mit *times*-maliger Wiederholung, um die Teilung um Faktor 3 für die Sendefrequenz zu ermöglichen.
In Zeile 5 wird lediglich ein spezielles Hardwareregister gelesen, damit das Register, dass das Auftreten des Taktsignals mitteilt, gelöscht wird.

Die Funktion com_wait() als Ganzes stellt also sicher, dass mit einer Zielfrequenz von $12\,kHz$ (und deren Teiler, insbesondere also auch mit $4\,kHz$) gearbeitet werden kann, sofern die Arbeiten zwischen zwei Taktsignalen nicht länger dauern.
Da jedoch in diesem Fall durch die Funktion nicht auf ein neues Taktsignal gewartet wird, wird eine Befehlsabarbeitung dann zumindest noch mit voller Prozessortaktgeschwindigkeit unterstützt.

Senden

Zum Senden werden nun hardwarenah folgende Funktionen zur Verfügung gestellt:

- com_send_init()

  Führt die Kommunikationsinitialisierung gemäß Kommunikationsprotokoll durch, initialisiert dabei auch den PIT.

- com_send_one() bzw. com_send_zero()

  Grundlegende Senderoutinen, sind in der Reihenfolge com_send_one() - Databit - com_send_zero() aufzurufen.

- com_send_end()

  Sendet abschließende Eins, damit der Empfänger eine letzte steigende Flanke bekommt, und deaktiviert den PIT, um Energie zu sparen.

Diese Funktionen werden gemäß Abbildung 3.36 aufgerufen.

Empfangen

Zum Empfangen werden hingegen folgende hardwarenahe Funktionen genutzt:

- com_rcv_init()

  Führt die Kommunikationsinitialisierung gemäß Kommunikationsprotokoll durch, initialisiert dabei auch den PIT.

- com_waitfor_posedge()

  Wartet auf den Sender, dessen Kommunikationsinitialisierung lt. Kommunikationsprotokoll länger dauert.

- com_rcv_bit()

  Wertet das Signal am Fototransistor mit 12 kHz von einer steigenden bis zur nächsten steigenden Flanke aus, entscheidet dann per Majoritätsentscheidung, ob eine “1„ oder eine “0„ empfangen wurde.

- calcBit()

  Berechnet zu den ersten sieben empfangenen Bits das Paritätsbit, um dieses auf höherer Ebene mit dem empfangenen Paritätsbit zu vergleichen.

Diese Funktionen werden gemäß Abbildung 3.35 aufgerufen.

Da jedoch beim Empfang von Nachrichten Fehler möglich sind, wird eine Fehlerdetektion wie folgt unterstützt:

- Timeout nach Initialisierung:

  Zum einen kann auf Seiten des Spielfeldes ein Problem auftreten, zum anderen kann der Roboter auf einem neuen Spielfeld ankommen und ihm implizit durch einen Verbindungsaufbau mitteilen, dass dieses die Navigations-LEDs ausschalten kann.
  In beiden Fällen folgt auf die Initialisierung keine Nachricht.

  Damit zum einen Energie gespart wird und zum zweiten eine erneute Initialisierung für eine spätere Kommunikation möglich ist, ist ein Timeout zu nutzen, um diesen Empfangsvorgang erfolglos abzubrechen.

- Timeout während des Empfangens von Bits:

  Dauert das Empfangen eines Bits zu lange, kann es sich nur um einen Kommunikationsfehler handeln, so dass eine Mehrheitsentscheidung nicht zwingend zum richtigen Ergebnis führen muss. Dementsprechend ist sicherheitshalber der Empfang abzubrechen, insbesondere da per Paritätsbit nur eine begrenzte Menge Fehler entdeckt werden kann.

- Falsches Paritätsbit:

  Trotz Sicherheitsvorkehrung im vorherigen Schritt ist ein Fehler aufgetreten, die empfangene Nachricht ist zu verwerfen.

Wird nun ein Fehler erkannt, so wird die Kommunikationsroutine beendet und liefert an die aufrufende Roboter-main Funktion eine Fehlermeldung.
Der Roboter reagiert daraufhin mit einer Kommunikationsfehlernachricht, so dass das Spielfeld weiß, dass zum einen evtl. eine Nachricht nicht angekommen ist, und zum zweiten der Roboter bereit für eine weitere Kommunikation ist[14].

3.3.7.5 Nachrichtenauswertung

Wird eine Nachricht korrekt empfangen, muss diese nun ausgewertet werden um die enthaltene Anweisung auszuführen. Dies geschieht in der main - Funktion (vgl. Abb. 3.33).

Dabei wird anhand der Nachrichtenidentifikationsbits der auszuführende Befehl decodiert und anschließend der zugehörige Parameter aus den Datenbits berechnet - z.B. die Anzahl der Felder bei einer Vorwärts- oder Rückwärtsbewegung.
Die Robotersoftware kennt neben Bewegungsbefehlen eine Identifikationsanforderung, die automatisch durch die Empfangsbestätigung bedient wird.

Es verbleibt, die Ausführung der Bewegungsbefehle näher zu betrachten.

[14] Dies ist insbesondere dann wichtig, wenn das Spielfeld den Roboter aufgefordert hat, das Spielfeld zu verlassen, da es sonst nicht weiß, dass der Roboter dennoch auf ihm stehen geblieben ist.

3.3.7.6 Motorsteuerung

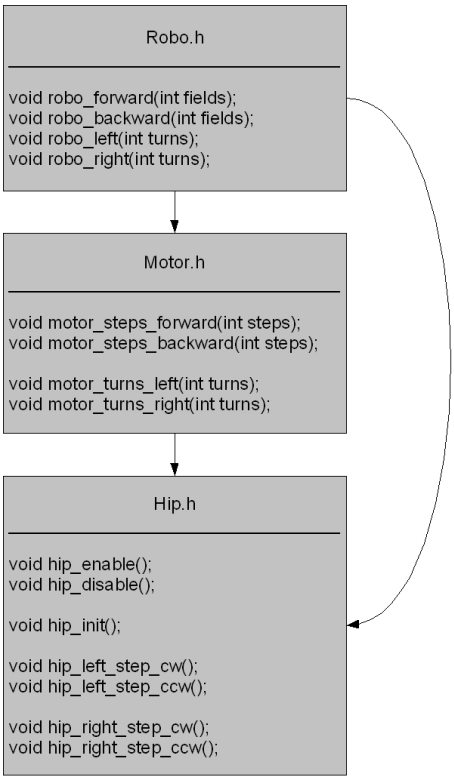


Abbildung 3.38: Motorsteuerung Funktionenübersicht

Die Bewegungen des Roboters, also genauer die Ansteuerungen der beiden Motoren, die den Roboter fortbewegen, werden in drei Ebenen realisiert (vgl. Abbildung 3.38):

Unterste Ebene: Ansteuerung der HIP4020 (Motortreiber)

Die einfachsten Funktionen, die bereitzustellen sind, müssen das Einschalten sowie das Ausschalten der HIPs - und damit der Motoren - ermöglichen. Da sich die Motortreiber eine Enableleitung teilen, ist lediglich diese über einen PIO-Pin anzusteuern. Das übernehmen die Funktionen hip_enable() und hip_disable().

Darüber hinaus werden für jeden der beiden Motoren (left und right benannt) Routinen gebraucht, die eine Drehung um genau einen Schritt im Uhrzeigersinn (clockwise: hip_<*Motorname*>_step_cw()) bzw. gegen den Uhrzeigersinn (counterclockwise: hip_<*Motorname*>_step_ccw()) erlauben.
Um diese realisieren zu können, ist die Ansteuerung der Motoren genauer zu betrachten [Nan04a]:

Zustand	Motoreingänge A	B	A'	B'	Motordrehrichtung
					CCW
1	+	+	-	-	
2	-	+	+	-	
3	-	-	+	+	
4	+	-	-	+	
					CW

Legende:
+ = Leitung an Versorgungsspannung anschließen
- = Leitung an GND anschließen

Abbildung 3.39: Motoransteuerungstabelle

Die vier verschiedenen Zustände (vgl. Abbildung 3.39) sind - je nach gewünschter Drehrichtung des Motors - von oben nach unten bzw. von unten nach oben zyklisch zu durchlaufen.
Da in jedem der vier Zustände A und A' bzw. B und B' genau invertiert zueinander sind, können sie je durch ein gemeinsames Signal angesteuert werden. Dieses ist lediglich für einen der beiden stets zu invertieren, was durch die zueinander inversen Eingänge der HIPs bereits erreicht wird.

Ergo reichen zwei Steuerleitungen - über zwei PIO-Pins angesteuert - pro Motor. Dabei wird durch die hip_<*Motorname*>_step_cw()- bzw. hip_<*Motorname*>_step_ccw()-Routinen die jeweils nächste Codierung an die PIO-Pins angelegt.
Hierfür ist der aktuelle Zustand zu speichern und allen Routinen zugänglich zu machen.

Der Notwendigkeit einer Initialisierung gerecht werdend (vgl. 3.3.3.2) ist darüber hinaus eine Routine vorhanden (hip_init()), die die Motortreiber in einen definierten Anfangszustand versetzt und durch Drehen der Räder beim Anschalten des Roboters selbiges auch für die Motoren sicherstellt.

Zwischenebene: Ansteuerung der Motoren

Um eine gleichmäßige Bewegung zu erhalten, müssten die Motoren parallel angesteuert werden.
Dies ist jedoch mit einem Mikrocontroller - ohne Programmcode oder Zustandsanzahl stark wachsen zu lassen - kaum möglich. Da zudem Korrekturen erschwert würden (komplexe Zustandswechsel wären notwendig oder Motoren nur gemeinsam ansteuerbar), wird eine fast parallele Lösung bevorzugt.

Dies geschieht durch im Wechsel unmittelbar nacheinander angesteuerte Motoren. Da der Mikrocontroller viel schneller arbeitet, als die Peripherie reagiert, und er nach einem Zustandswechsel Pausen einlegen muss, um die Motoren nicht zu über-

lasten, wird durch diese fast parallele, abwechselnde Motoransteuerung für den Menschen trotzdem eine gleichmäßige Bewegung wahrnehmbar.

Genau dies setzen die Funktionen der Zwischenebene unter Angabe einer Motordrehschrittanzahl um.

Oberste Ebene: Steuerung des Roboters

Auf der obersten Ebene wird der Roboter als Einheit gesteuert. Dazu werden Prozeduren zur Verfügung gestellt, die echte spielelementare Bewegungen ermöglichen. Gemäß der Spielanforderungen sind dies Routinen zur:

- Vorwärtsbewegung,
- Rückwärtsbewegung,
- Linksdrehung sowie
- Rechtsdrehung.

Um alle Spielzüge hiermit direkt umsetzen zu können, wird den Routinen ein Anzahlparameter übergeben. Spielzüge à la „zwei Felder vor“ lassen sich alternativ aus zwei elementaren „ein Feld vor“ Anweisungen zusammensetzen usw.

Diese Entscheidung bietet zudem größte Flexibilität bzgl. der Nutzung und Aktivierung von Fehlerkorrekturalgorithmen wie in 3.3.7.7 beschrieben. Sie kann so etwa an jeder LED eines Spielfeldes, an jedem Spielfeld oder gar erst zum Ende jedes Spielzuges zum Einsatz kommen.
Es sind verschiedene Korrekturen denkbar, z.B. kleinere Korrekturen an den LEDs während der Fahrt, größere erst an der LED zum Ende des Spielzuges.

Abgesehen von der Fehlerkorrektur ist die Umsetzung der Bewegung trivial. Es sind lediglich die entsprechenden Funktionen der Zwischenebene mit der zu erwartenden Schrittanzahl aufzurufen, welche im folgenden Abschnitt näher untersucht wird. Hinzu kommen Fehlerkorrekturaufrufe an passenden Stellen, ferner das Einschalten der HIPs zum Beginn und das Ausschalten zum Ende der Prozedur.

Parameter

Die Anzahl der Motorschritte, die der Roboter für eine Vierteldrehung oder für die Bewegung um ein Feld (bzw. $\frac{1}{3}$ Feld bei Korrekturen an jeder Weg-LED) weiter benötigt, müssen abgeschätzt werden.

Sofern keine unvorhersehbaren äußeren Einflüsse die Bewegung verfälschen (z.B. Rutschen beim Anfahren) ist eine exakte Berechnung möglich, weil alle Parameter wie Raddurchmesser, Radabstand, Motordrehwinkel, Getriebeuntersetzung und Entfernungen bekannt sind.

Durch empirische Beobachtungen kann diese Berechnung den realen Gegebenheiten angepasst werden, um so für eine Fehlerkorrektur eine gute Ausgangssituation zu schaffen.

Zunächst sind alle Daten zu sammeln:

- Motordrehwinkel: 18°
- Getriebeuntersetzung: 30 : 1
- Raddurchmesser: $12,25\,mm$
- Radabstand: $33,9\,mm$
- Feldabstand: $50\,mm$
- Drehwinkel: 90°

$$Raddrehwinkel = Motordrehwinkel * Getriebeuntersetzung$$

$$Raddrehwinkel = 18° * \frac{1}{30} = \frac{9}{15}^{\circ} \tag{3.1}$$

$$Kreisbogen\ Rad = \frac{2 * \pi * Radradius * Raddrehwinkel}{360°}$$

$$Kreisbogen\ Rad = \frac{2 * \pi * \frac{12{,}25\,mm}{2} * \frac{9}{15}^{\circ}}{360°} \approx 0,0641\,mm \tag{3.2}$$

$$Schritte\ pro\ Feld = \frac{Feldabstand}{Strecke\ pro\ Motorschritt}$$

$$Schritte\ pro\ Feld = \frac{50\,mm}{0,0641\ \frac{mm}{Schritt}} \approx 780\,Schritte \tag{3.3}$$

Die Drehung eines Motors bewegt nach Untersetzung mit einem Motorschritt ein Rad um $\frac{9}{15}^{\circ}$. Der Kreisbogen des Rades - und damit die Strecke, die pro Motorschritt gefahren wird - beträgt so $0,0641\,mm$.
Entsprechend sind für eine Strecke von $50\,mm$ etwa $780\,Schritte$ des Motors notwendig.

$$Kreisbogen\ Robo = \frac{2 * \pi * halber\ Radabstand * Drehwinkel}{360°}$$

$$Kreisbogen\ Robo = \frac{2 * \pi * \frac{33{,}9\,mm}{2} * 90°}{360°} \approx 26,625\,mm \tag{3.4}$$

$$Schritte\ pro\ Drehung = \frac{Strecke}{Strecke\ pro\ Motorschritt}$$

$$Schritte\ pro\ Drehung = \frac{26,625\,mm}{0,0641\ \frac{mm}{Schritt}} \approx 415\,Schritte \tag{3.5}$$

Eine Drehung von 90° des Roboters bedeutet eine zu bewältigende Strecke von $26,625mm$ und erfordert so 415 *Schritte*.

Eine empirische Überprüfung und Korrektur (z.B. für einen regelmäßigen Schlupf) dieser Werte steht noch aus.

3.3.7.7 Fehlerkorrekturalgorithmus

Eine ganze Reihe unberechenbarer Faktoren wirken auf Spielfeld und Roboter ein - zum Beispiel ein Rutschen der Räder auf dem Makrolon oder Verunreinigungen, die die Bewegung stören - und bewirken so, dass die Bewegung des Roboters nicht ganz genau vorhergesagt werden kann.

Alle Fehler für sich genommen sind sehr klein, aber in Summe und vor allem über ein komplettes Spiel hinweg gesehen können sie das Spielvergnügen erheblich beeinträchtigen, wenn der Roboter einmal nicht mehr an seinem Ziel ankommt, mit einem anderen Roboter vielleicht kollidiert oder gar von der Spielfläche stürzt. Folglich ist es notwendig, diese Fehler zu korrigieren.

Ziel

Ziel ist, dass nach der Korrektur der Roboter genau platziert ist, darüber hinaus sollte sich aber diese Korrektur möglichst in den Spielfluss integrieren.

Lang dauernde Korrekturzüge oder große Bewegungen führen zu einer Spielverzögerung und für den Benutzer nicht nachvollziehbaren Bewegung der Roboter und reduzieren so den Spielspaß erheblich.
Ferner steht nur begrenzter Platz für die Korrekturzüge zur Verfügung, da das aktuelle Spielfeld nicht verlassen werden darf, denn ein Nachbarfeld kann durch einen zweiten Roboter besetzt sein.

Diese gegenläufigen Ziele aus Präzision und Unauffälligkeit gilt es geschickt zu vereinen.

Fehler

Am Anfang des Korrekturalgorithmusentwurfs steht die Fehleranalyse.
Dabei werden zwei Situationen unterschieden, Bewegungen und Drehungen, da die Fehlerursachen und -auswirkungen sich dabei in wesentlichen Punkten unterscheiden.

Drehungen sind relativ unproblematisch, da der Roboter sich nicht von seinem Feld fortbewegen muss. Somit hat über die gesamte Drehung stets der mittlere Fototransistor Sichtkontakt zur zentralen LED des Feldes, um die Position des Roboters zu kontrollieren. Zusätzlich kann mittels des vorderen Fototransistors auch die Blickrichtung des Roboters anhand der Navigations-LEDs im Feld überprüft werden.

Bei der geraden Fortbewegung hingegen sind eine ganze Reihe von Fehlersituationen vorstellbar:

- der Roboter ist zu kurz gefahren (z.B. durch durchdrehende und/oder rutschende Reifen),
- der Roboter ist nach links oder rechts abgewichen (ungenaue vorherige Ausrichtung oder einseitig stärkerer Schlupf), oder
- der Roboter ist zu weit gefahren (eher unwahrscheinlich).

Dabei muss beim Entwurf davon ausgegangen werden, dass sich die Fehler in begrenzten Rahmen bewegen, da sonst keine effiziente Korrektur möglich ist. So könnte nicht garantiert werden, dass keine dritten Felder befahren würden; nicht einmal, dass das Zielfeld überhaupt gefunden würde.

Folgende Annahmen bestimmen den Entwurf und gelten pro 90° Drehung bzw. pro Fortbewegung um $\frac{1}{3}$ Feld ($\frac{50}{3}\,mm$). Die Berechnungen basieren auf der Parameterberechnung in 3.3.7.6.

- Im Winkel wird maximal um 5° abgewichen.

  Dies entspricht bis zu 23 Schritt-Fehlern beider Motoren, gar deutlich mehr einseitigen Schrittaussetzern in der geraden Bewegung.

- In der Strecke wird maximal um $1\,mm$ abgewichen.

  Dies entspricht über 15 beidseitigen Aussetzern in der geraden Bewegung.

Dies erscheint recht großzügig, da aber Fehler nur in bestimmten Maßen behoben werden können (unter anderem durch begrenzte Messgenauigkeit über die Fototransistoren), sind diese Werte bewusst relativ hoch angesetzt.

Eine Suche kann unter diesen Annahmen nun innerhalb dieser Grenzen erfolgen. Falls der Fehler sich bei der Suche als größer erweisen sollte, ist sie erfolglos abzubrechen.
In diesen Fällen muss der Benutzer eingreifen und manuell korrigieren. Dies muss jedoch eine Ausnahmesituation bleiben!

Idee

Grundlage für die Korrektur nach einer vorausberechneten Drehung (vgl. 3.3.7.6) ist eine Suche durch weitere kleinerer Links-Rechts-Drehungen wie Abb. 3.40 verdeutlicht.

So wird der Roboter gerade positioniert und steht für den nächsten Spielzug bereit.

Die Korrektur für die schwerwiegenderen Fehler der geraden Fortbewegungen setzt sich aus zwei Teilen zusammen:

- einer leichten Korrektur auf dem Weg an jeder LED, um den Fehler gering zu halten und das Erreichen des Zieles sicher zustellen, sowie

```
Solange Lichtintensitaet nicht sinkt:
  Drehe links;
Merke linkes Maximum;
Solange Lichtintensitaet nicht sinkt:
  Drehe rechts;
Merke rechtes Maximum;
Positioniere Roboter auf Mitte
  zwischen linkem und rechtem Maximum;
```

Abbildung 3.40: Idee für Drehkorrektur (Pseudocode)

- einer finalen Korrektur auf jedem Spielfeld, wobei hier je nach Schwere der Abweichung sowohl eine leichte Korrektur als auch eine starke Korrektur mit entsprechend spezialisierten Algorithmen denkbar ist. Letztere kann größere Fehler sicherer und schneller beheben, wird aber für den menschlichen Spieler wesentlich auffälliger agieren.

Durch ihren gemeinsamen Einsatz soll möglichst unauffällig sichergestellt werden, dass das nächste Spielfeld sicher erreicht wird. Dabei sehen die Konzepte konkret wie folgt aus:

intermediate_correction: Die leichte Korrektur an den Weg-LEDs

Falls der Roboter in eine falsche Richtung läuft - genauer gesagt leicht vom Kurs nach links oder rechts abweicht - soll der Kurs korrigiert werden, um den Fehler nicht größer werden zu lassen, ihn vielleicht gar zu verkleinern. Dafür kann nach der vorausberechneten Vorwärtsbewegung[15] analog zum Drehalgorithmus nach Abb. 3.40 die Suche nach maximaler Lichtintensität aufgenommen werden. In ihre Richtung wird dann korrigiert. Ein Beispiel weiter unten verdeutliche dies.

final_light_correction: Die sanfte finale Korrektur

Sie soll dem Prinzip des Einparkens eines Pkws folgen, dieser Gedanke muss jedoch noch weiter entwickelt werden.

final_heavy_correction: Die große finale Korrektur

Diese Korrektur vermag zwar große Fehler zu berichtigen, allerdings stellt sie auch einen großen Eingriff in den Spielablauf dar, da der Spielroboter unter anderem zwei Vierteldrehungen durchführen muss, um so in beiden Spielfelddimensionen nach dem Maximum der Lichtintensität zu suchen. Auch dies verdeutliche ein Beispiel weiter unten.

Sobald die beiden finalen Korrekturen entwickelt und implementiert sind, ist zu entschei-

[15] Rückwärtsbewegungen funktionieren analog, es wird wegen des Messpunktes hinten lediglich in die andere Richtung korrigiert werden.

den, in welcher Situation welche Variante zum Einsatz kommt. Ein mögliches Entscheidungskriterium liefert hierbei die auf dem Weg auftretenden Fehler, die unter Umständen eine schwerwiegendere Korrektur notwendig machen. Dies lässt sich am Einsatz der intermediate_correction erkennen.
Dementsprechend kann, falls eine Korrektur nötig war, die heavy_correction ausgeführt werden, sonst wird die light_correction ausreichen.

Beispiel intermediate_correction

Anhand zweier Fehlersituationen mit einer Drehungabweichung von 5° zur Zielrichtung soll die intermediate_correction demonstriert werden.

Diese werden als Extrema gewählt, nämlich einmal mit der frühestmöglichen Abweichung sofort am Start (Fall $\mathcal{A}$) und einem dementsprechenden Lauf in die falsche Richtung - dargestellt in den Abbildungen 3.41 bis 3.47 - und einer spätestmöglichen Abweichung bei der Ankunft auf der nächsten LED (Fall $\mathcal{B}$) - dargestellt in den Abbildungen 3.48 und 3.49 -, die so alle Fehler dieser Kategorie umfassen.

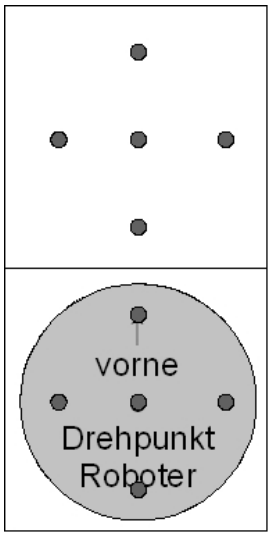


Abbildung 3.41:
$\mathcal{A}$ - Ausgangssituation

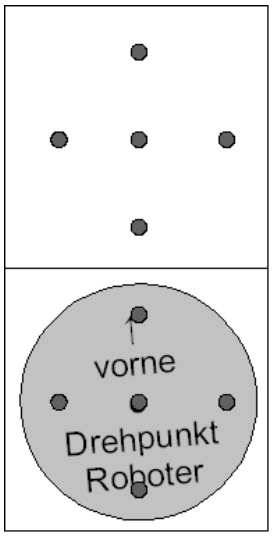


Abbildung 3.42:
$\mathcal{A}$ - Fehlersituation

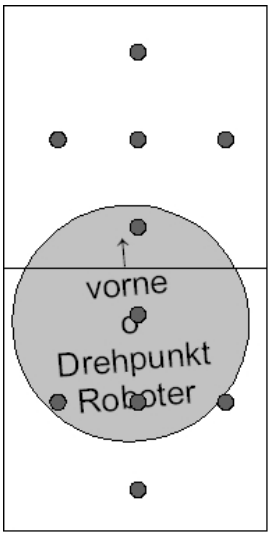


Abbildung 3.43:
$\mathcal{A}$ - Folgefehler

Fehler zu Beginn der Bewegung (Fall $\mathcal{A}$)

Ausgegangen wird von einer falschen Startposition bzw. einer Drehung direkt zu beginn der Bewegung (Abb. 3.42).
Durch Bewegung um $\frac{50}{3}\,mm$ vorwärts weicht der Roboter nun nicht nur zur Seite von seinem Ziel ab, er bleibt auch vor der nächsten LED stehen. Mittels Kosinussatz und Sinussatz ergeben sich Fehler von seitlich etwa $1,3\,mm$ und in der Länge von $1,6\,mm$ (Abb. 3.43).

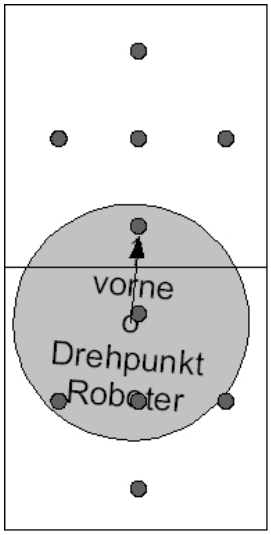


Abbildung 3.44:
$\mathcal{A}$ - Korrekturausrichtung

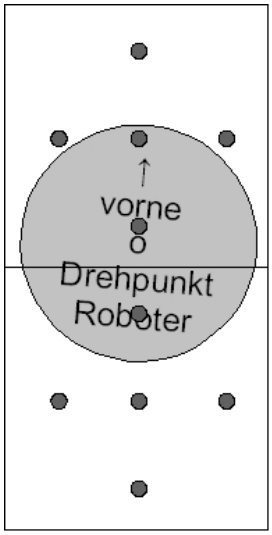


Abbildung 3.45:
$\mathcal{A}$ - Nächster Zug

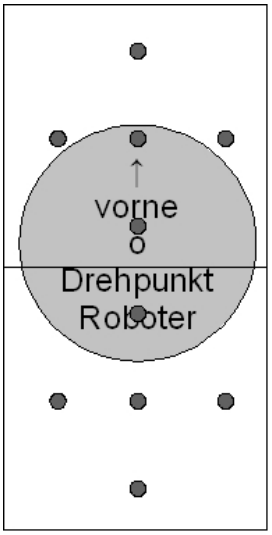


Abbildung 3.46:
$\mathcal{A}$ - Weitere Korrektur

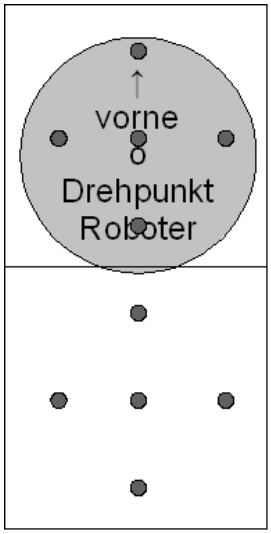


Abbildung 3.47:
$\mathcal{A}$ - Letzter Zug

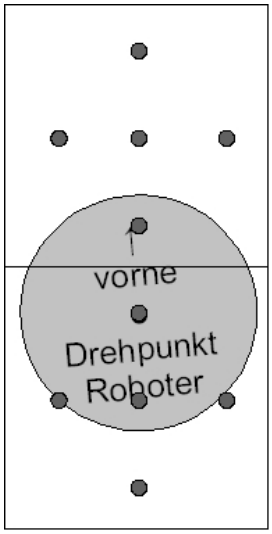


Abbildung 3.48:
$\mathcal{B}$ - Kleiner Fehler

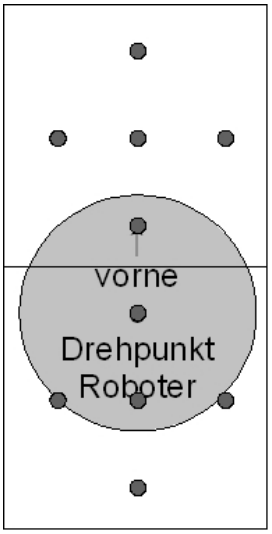


Abbildung 3.49:
$\mathcal{B}$ - Dessen Korrektur

Die Position wird in Richtung der LED korrigiert (Abb. 3.44), anschließend eine weitere Strecke von $\frac{50}{3}$ mm geradeaus gefahren (Abb. 3.45) und wiederum eine Korrektur durchgeführt (Abb. 3.46).
So ist der Roboter wieder korrekt ausgerichtet und kommt nach dem nächsten Zug auf dem Zielfeld an (Abb. 3.47). Lediglich eine Längenabweichung von kumuliert $3,3\,mm$ verbleibt. Diese ist in der finalen Korrektur zu beheben.

Alternativ bestünde auch die Möglichkeit, diesen Wert durch Anpassung der Laufstrecke von $\frac{50}{3}\,mm$ um einen Korrekturfaktor in Abhängigkeit zum Korrekturwinkel zu verringern. Eine exakte Korrektur durch Berechnung mittels trigonometrischer Funktionen kommt jedoch aus Effizienzgründen im Roboter nicht in Frage (um Speicherplatz zu sparen, stehen derartige C-Funktionen nicht zur Verfügung).

Fehler zum Ende der Bewegung (Fall $\mathcal{B}$)

Eine Vorwärtsbewegung um $\frac{50}{3}\ mm$ wird korrekt durchgeführt und die Ziel-LED erreicht. Erst im letzten Moment tritt der Fehler ein (Abb. 3.48). Dieser Fehler wird durch den Korrekturalgorithmus sofort vollständig behoben (Abb. 3.49).

Die Skizzen suggerieren die Toleranz auch höherer Fehler. Diese werden aber kritisch, wenn der Roboter sein Spielfeld verlässt, da er zum einen mit einem anderen Roboter und zum anderen mit einer Markierungsfahne der Spielfelder[16] kollidieren könnte. Dies geschieht schon ab einem kumulierten Fehlerwinkel von $7,75°$, da dann die seitliche Abweichung $2,25\,mm$ erreicht (Sinusdefinition) und eine Spielfeldgrenzverletzung droht.

Beispiel heavy_correction

An einem einfachen Fehlerfall einer beliebigen Abweichung gemäß der Annahmen der Fehlerabschätzung oben (Fall $\mathcal{C}$) wird die starke Endkorrektur demonstriert.

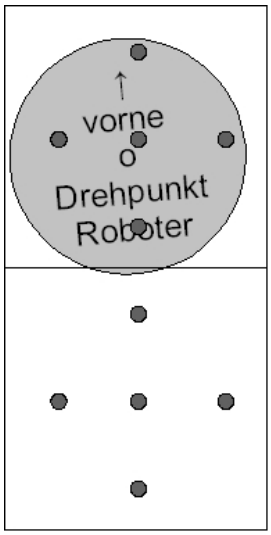


Abbildung 3.50:
$\mathcal{C}$ - Fehlersituation

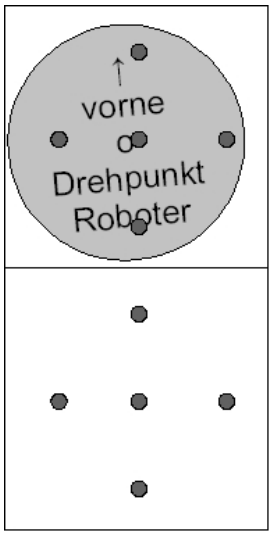


Abbildung 3.51:
$\mathcal{C}$ - 1. Maximumssuche

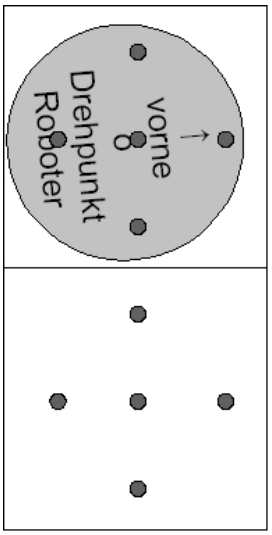


Abbildung 3.52:
$\mathcal{C}$ - Drehung

[16] Die Markierungsfahnen, die Checkpoints markieren, werden in 3.4.4.3 erläutert. Hier genügt es zu wissen, dass Fähnchen am Rande der Spielfelder stehen können.

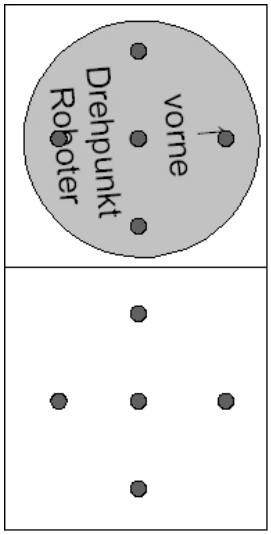


Abbildung 3.53:
$\mathcal{C}$ - 2. Maximumssuche

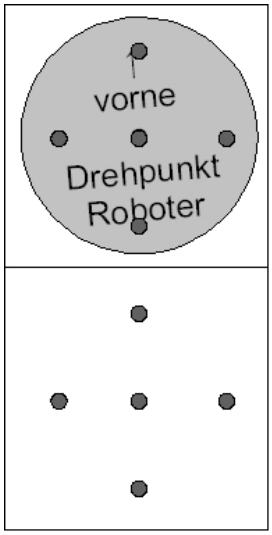


Abbildung 3.54:
$\mathcal{C}$ - Inverse Drehung

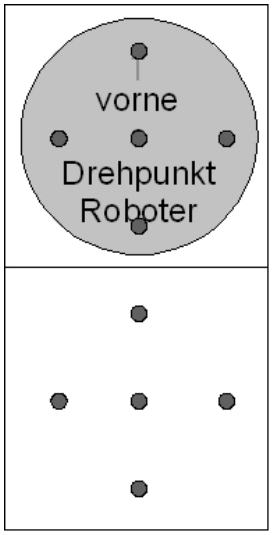


Abbildung 3.55:
$\mathcal{C}$ - Drehkorrektur

Das Vorgehen folgt folgendem Schema:

Nach der vorherberechneten Bewegung des Roboters (Abb. 3.50) sucht dieser nach der maximalen Lichtintensität in Laufrichtung. Er kommt so im minimalen Abstand zur Positionierungs-LED zu stehen (Abb. 3.51).
Nach einer Drehung um 90° (Abb. 3.52) erfolgt eine zweite Positionierung an der LED. Durch diese 2-dimensionale Suche ist sichergestellt, dass der Roboter nun unmittelbar auf der Positionierungs-LED steht (Abb. 3.53), da die erste Korrektur der Suche des Berührpunktes eines Kreises um die LED auf einer Tangente entspricht, die zweite Suche einer Kreismittelpunktsuche entlang des Tangentialradius.
Nach der inversen 90° Drehung (Abb. 3.54) muss eine Korrektur der Roboterausrichtung erfolgen. Dies geschieht vollkommen analog zur Drehkorrektur gemäß des Algorithmus in Abb. 3.40).

Ergebnis

Die Qualität der Fehlerkorrektur hängt ganz wesentlich von mehreren Faktoren ab:

- die Qualität - d.h. Genauigkeit - der Lichtintensitätsmessung,
- die Abschätzung der Fehlersituationen,
- die Abschätzung der Fehlerschwere sowie
- das Zusammenspiel aller Hardwarekomponenten.

Letztendlich kann nur die Praxis zeigen, ob Modell und Lösung den Anforderungen und den wahren Begebenheiten entsprechen können.

3.3.7.8 Softwareverifikation

Die Verifikation der Software im Roboter erweist sich als sehr schwierig:

Zum Einen sind viele Fehlerquellen möglich.
So könnte eine Hardwarekomponente nicht fehlerfrei arbeiten, das Zusammenspiel zwischen Komponenten nicht funktionieren, die Interaktion zur Umwelt fehlschlagen, eine Mikrocontrollerkomponente nicht wie gewünscht arbeiten oder letztendlich ein simpler Programmierfehler vorliegen. Dementsprechend ist ein Fehlverhalten des Roboters verbunden mit einer aufwendigen Suche nach der möglichen Ursache.
Ohne auf einzelne Fehler einzugehen, seien hier ein paar Beispiele erwähnt: Auf dem Makrolon rutschende Reifen, schwer anzusteuernde Interrupts des Mikrocontrollers, Kurzschlüsse durch Überbrückung von Widerständen über Motorgehäuse oder einfache Speicherzugriffsfehler.

Auf der anderen Seite wird die Softwareverifikation durch die begrenzten Aktionsmöglichkeiten des Roboters erschwert. So ist es bei einem Kommunikationstest nicht einmal möglich mit bordeigenen Mitteln festzustellen, ob das Programm überhaupt noch arbeitet, geschweige denn an welcher Stelle sich die Programmabarbeitung befindet.

Letztendlich reagiert der Roboter zudem nur auf externe Reize, die ganz speziellen Mustern zu folgen haben. Insofern ist der Roboter im Test abhängig von einer weiteren, nicht vollständig allein testbaren, komplexen Komponente: dem Spielfeld.

Da jedoch der Test des Roboters vor allem durch den Test der Kommunikation bestimmt ist, geht dieser vollständig im Integrationstest auf. Nur der Mikrocontroller selbst sowie einige Programmteile und deren Komponenten lassen sich im Vorfeld testen, insbesondere der Antrieb.

Mikrocontrollerkomponenten

Zum Test des Mikrocontrollers steht ein Testboard wie in 3.3.4 beschrieben zur Verfügung. So können alle Komponenten des Mikrocontrollers unter Nutzung vieler Ausgabemöglichkeiten in Betrieb genommen werden, zum Beispiel der ADC (Analog/Digital Converter), der PIT (Periodic Interval Timer) oder der PIO (Parallel Input/Output Controller).

Antriebstest

Für einen Antriebstest können sukzessive sämtliche Routinen von einer Testprozedur aus aufgerufen werden. An einem über zwei HIP angeschlossenen Motor lässt sich die

Funktionalität sofort erkennen.
So konnten von den Routinen zur Drehung des Motors um einen einzelnen Schritt bis hin zu den Routinen, die den Roboter vorwärts, rückwärts oder im Kreis laufen lassen, alle für den Roboterbetrieb relevanten Routinen ausgiebig getestet werden.

Weiterführende Tests, d.h. vor allem der Test der Kommunikation und der Test der main.c Funktion, fanden als Integrationstests beschrieben in Abschnitt 3.5 statt.

3.3.7.9 Fazit

Obwohl im Roboter keine Spielintelligenz zu implementieren, sein Funktionsumfang stark begrenzt und deren Komplexität überschaubar ist, erweist sich die Programmierung des Roboters - insbesondere die folgende Testphase - als schwierig, da an dieser Stelle alle Fäden der Roboterkonstruktion zusammenlaufen. Um so erfreulicher ist es, wenn der Roboter erstmals seinen Dienst verrichtet.

3.3.8 Ergebnisse

In Abbildung 3.56 kann man sehen, wie das Ergebnis des Roboterprototypen, zusammengesetzt aus sämtlichen oben beschrieben Elementen, aussieht. Die vorangegangenen Abschnitte sollten den Aufbau des Roboterprototypen und damit die baldige Fertigung der Serienroboter erläutern. Es wird deutlich, dass strenge Anforderungen bezüglich der räumlichen und technischen Gegebenheiten an den Roboter gestellt werden. In dem folgenden Kapitel wird darauf eingegangen, wie das Spielfeld, das mit dem Roboter perfekt harmonieren und kommunizieren muss, aufgebaut ist.

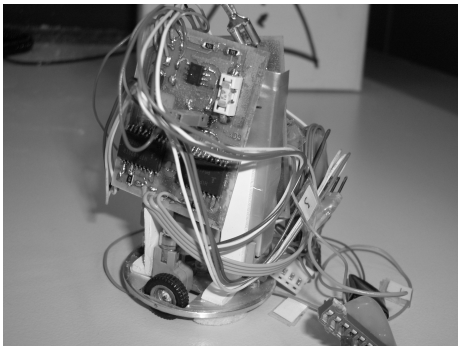


Abbildung 3.56: Der komplette Roboterprototyp

3.4 Spielfeld

3.4.1 Allgemeiner Aufbau

3.4.1.1 Anforderungen

Die Spielpläne des Brettspiels RoboRally setzen sich aus einzelnen Feldern zusammen. Diese einzelnen Felder wird es auch in der R_3D_3 Realisierung geben. Hier werden 9 dieser Felder zu einer so genannten Spielfeldbox zusammengefasst. Eine Spielfeldbox besteht dabei aus 3 x 3 Feldern. In diesem Teil des Zwischenberichts werden hauptsächlich die Spielfeldboxen betrachtet und nicht einzelne Felder. In dem PG-Antrag steht die Vorgabe, dass die einzelnen Spielfelder 5cm x 5cm groß sein sollen, dementsprechend wird jede Spielfeldbox 15cm x 15cm groß sein.

Eine weitere Anforderung an die Boxen ist, dass alle Boxen auf beliebige Weise miteinander kombinierbar sein sollen. Das bedeutet, dass jede Box beliebig gedreht werden kann und trotzdem alle Boxen zusammenpassen. Damit jede Box an jede andere Box gesteckt werden kann, müssen alle Seiten der Spielfeldbox gleich aussehen. Dieses wird vor allem in dem Kapitel über die Kontakte noch einmal näher betrachtet, siehe Kapitel 3.4.5.2.

Über die Boxen müssen die Roboter fahren können. Deshalb werden die Spielfeldboxen eine Oberfläche haben, die dieses ermöglicht, näheres hierzu in Kapitel 3.4.4.

Damit die Roboter “gerade“ über die Boxen fahren können, werden Navigationshilfen benötigt. Dank diesen kann der Roboter exakt von einem zu nächsten Feld fahren. Als Navigationshilfen werden LEDs benutzt, näheres hierzu in Kapitel 3.4.6 und Kapitel 3.3.7.7.

Bevor nun die konkrete Realisierung der Spielfeldbox beschrieben wird, werden hier zunächst die Anforderungen an das Material genannt:

1. Über die Spielfeldbox müssen später die Roboter fahren, dementsprechend muss die Box stabil genug sein, um den Roboter tragen zu können. Daraus resultiert, dass als Material für die Box nur Holz, Plastik und Metall verwendbar sind. Andere evtl. mögliche Materialien (wie bpsw. Pappe, Styropor, ...) müssen somit nicht mehr betrachtet werden.

2. Das Material der Spielfeldbox sollte sich nicht durch Umwelteinflüsse wie bspw. eine zu hohe Luftfeuchtigkeit verändern. Dies ist ein eindeutiger Nachteil von Holz, da sich Holz bei einer zu hohen Luftfeuchtigkeit verziehen kann.
   So könnte es passieren, dass die Spielfeldbox nicht mehr exakt rechtwinklig ist. Wenn nun mehrere solcher Boxen aneinandergesetzt werden, könnten Zwischenräume entstehen. Der Roboter müsste somit eventuell größere Zwischenräume zwischen den Boxen überwinden.
   Außerdem könnte es passieren, dass die Ecken der Box nicht mehr in einer Ebene liegen. Somit würde die Spielfeldbox wackelig auf dem Tisch stehen.

3. Bei der Wahl von Metallen muss berücksichtigt werden, dass diese oxidieren. Dies schließt Metalle wie z.B. Eisen, aus. Eventuell mögliche Metalle sind Stahl und Aluminium.

4. Wie in Kapitel 3.4.5 beschrieben, soll mit Magneten die Stabilität des Gesamtspielfeldes sichergestellt werden. Da hierbei Magnete gegensätzlich gepolt werden, muss das Material magnetisch neutral sein. Dies schließt Metalle wie z.B. Eisen aus.

5. Da die elektronischen Bauteile während des Betriebs Wärme erzeugen können, sollte das Material eine gute thermische Leitfähigkeit aufweisen können. Im Gegensatz zu Holz und Plastik hat Metall obige Eigenschaft.

6. Das Material sollte sich mit den vorhandenen Werkzeugen des Lehrstuhls 12 bearbeiten lassen. Am Lehrstuhl sind geeignete Maschinen für die Metallverarbeitung vorhanden. Mit diesen Maschinen kann Metall mit einer hohen Genauigkeit bearbeitet werden. Da Holz oder Plastik nur mit einer Handsäge bearbeitet werden kann, lässt sich hier nur eine wesentlich geringere Genauigkeit erzielen.
   Auch hier könnten wieder Zwischenräume entstehen, die der Roboter überwinden müsste.

3.4.1.2 Parkplatz

In der deutschen Ausgabe des Spiels starten alle Roboter auf einem Feld, siehe [AMI99]. Dies wird so umgesetzt, dass nacheinander die Spieler in Reihenfolge der Prioritäten auf den Programmkarten die Roboter auf das Feld setzen. Für die R3D3 Realisierung ist das nicht die optimale Lösung. Deswegen wird an dieser Stelle teilweise auf die englische Spielversion zurückgegriffen. In dieser Version existiert ein gesonderter Startspielplan [Wiz05]. Das bedeutet, dass am Anfang jedes Spiels jeder Roboter auf seinem ausgewiesenen Spielfeld stehen kann und so von einem anderen Spielfeld startet. So gibt es nicht das Problem, dass jeder Spieler seinen Roboter zu einem bestimmten Zeitpunkt auf dem Spielfeld platzieren muss. Die Ausführung des Programms muss also in keiner Weise vom Spieler durchgeführt werden. Allerdings wird auf diese Weise ein Teil der Anfangsspannung aus dem Spiel genommen. Deshalb können die Vorteile beider Varianten folgendermaßen miteinander kombiniert werden: Es gibt einen gesonderten Parkplatz für die Roboter (siehe Abb. 3.57). Auf diesem werden am Anfang des Spiels die Roboter platziert. Dabei wird jeder Roboter gemäß seiner Spieler ID auf dem Feld mit der jeweiligen Ziffer platziert. Außerdem gibt es noch den richtigen Startpunkt für das Spiel. Nachdem die Spieler ihr jeweiliges Programm ausgewählt haben, sorgt das Spielfeld dafür, dass die Roboter zum richtigen Zeitpunkt vom “Parkplatz“ zum Startpunkt des Spieles gelangen. Hierbei kann das Spielfeld den Roboter natürlich auch durch alle Elemente lenken, durch die ein Roboter im Spielverlauf normalerweise nicht fahren kann (bspw. Wände, Gruben). Außerdem kann dieser Parkplatz auch genutzt werden, wenn ein Roboter in eine Grube gefallen ist oder auf eine andere Weise “gestorben“ ist. Auf diese Weise muss der Spieler nur in das Spielgeschehen eingreifen, indem er sein Programm dem Host bzw. der Konsole mitteilt.

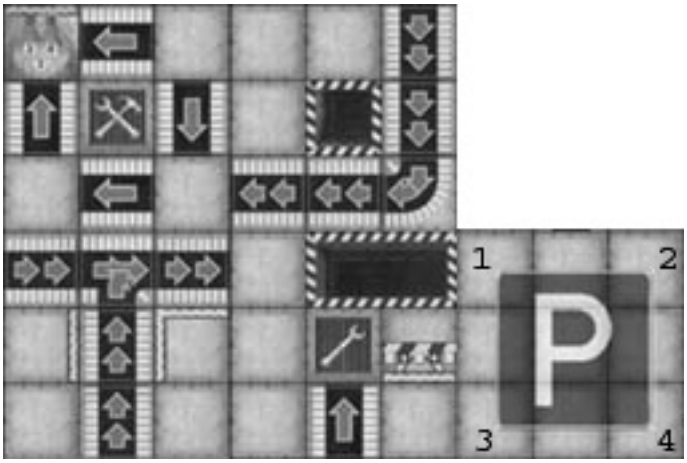


Abbildung 3.57: Spielfläche mit Parkplatz

3.4.2 Spielfeldbox - Material

Wie schon in 3.4.1.1 Punkt 1. gesehen, müssen nur noch die Materialien Plastik, Holz und Metall näher betrachtet werden:

1. **Plastik:**
   Eine reine Plastiklösung ist für nicht verwendbar, da dieses Material keine thermische Leitfähigkeit besitzt.
   Wie oben schon erwähnt, kann Plastik mit den am Lehrstuhl vorhandenen Werkzeugen nicht genau genug bearbeitet werden. Dieses Problem besteht besonders beim exakten Zuschneiden des Materials. Dies ist ein weiterer Nachteil von Plastik.

2. **Holz:**
   Holz hat den Vorteil, dass es einfach zu bearbeiten ist. Hier ist eine handelsübliche Säge und ein Bohrer zur Bearbeitung ausreichend.
   Nachteile von Holz sind wie oben schon genannt zum Einen die durch fehlende Maschinen nicht zu erreichende Genauigkeit beim Bearbeiten. Zum Anderen kann es passieren, dass sich Holz verzieht. Außerdem hat Holz eine zu geringe thermische Leitfähigkeit.
   Somit genügt Holz nicht unseren Ansprüchen.

3. **Metall:**
   Im Gegensatz zu Holz hat Metall den Vorteil, dass es elektrisch und thermisch leitfähig ist. Ein für uns besonders gut geeignetes Metall ist Aluminium, da es zusätzlich magnetisch neutral und relativ einfach zu bearbeiten ist. Ferner hat Aluminium den Vorteil gegenüber anderen Metallen, dass sich eine Oxidationsschicht an der Oberfläche bildet. Dies verhindert weiteres Oxidieren.

Unabhängig vom gewählten Material können entweder fertige Boxen gekauft werden, oder die Boxen selbst hergestellt werden.
Ein wichtiger Vorteil von fertig gekauften Boxen besteht darin, dass der große Aufwand der Selbstherstellung erspart bleibt. Doch leider bestehen auch einige Probleme: Die Boxen müssten noch weiter modifiziert werden. Zum Einen müssten Löcher für die in Kapitel 3.2.3 beschriebenen Kommunikationeinrichtungen gebohrt werden, und zum Anderen müssten Möglichkeiten für die Steckverbindungen zwischen den Boxen eingebaut werden. Zwischen den Boxen müsste es so einen Spalt für die Steckverbindungen geben. So wären wieder Zwischenräume vorhanden, die der Roboter überwinden müsste. Ein weiterer gravierender Nachteil von fertig gekauften Boxen ist, dass diese in der Regel runde Kanten haben. Dies würde für die Roboter ein nicht überwindbares Hindernis darstellen. Zusätzlich gab es keine Boxen der Größe 15cm x 15cm.

Daraus resultiert die Entscheidung, die Boxen selbst herzustellen. So können die Boxen nach individuellen Bedürfnissen erstellt werden.
Die Spielfeldbox wird in eine Unterkonstruktion und deren Oberfläche unterteilt. Beides wird gesondert in den folgenden Kapiteln betrachtet.

3.4.3 Unterkonstruktion

Die Unterkonstruktion besteht im Wesentlichen aus Seitenflächen. Es gibt mehrere Möglichkeiten um, die Oberfläche und die Seitenflächen zu verbinden:

1. Eine Idee ist, Aluminiumblöcke zu benutzen. In diese werden benötigte Gewinde für Schrauben zur Befestigung der Oberfläche und Seitenflächen geschnitten.
   Um Platz für die benötigten Kontakte zwischen den Boxen zu haben, dürfen die Seitenflächen nicht von außen gegen den Block geschraubt werden. Es bleibt nur noch eine Konstruktion, in der die Seitenflächen in den Block eingelassen werden.
   Wie später in Kap 3.4.5 beschrieben, sollen die Boxen über Magnete zusammengehalten werden. Bei dieser Lösung sind die Magnete von der Außenseite in den Block versenkt. Sie sind von der Außenseite sichtbar. Beim Aufbau des kompletten Feldes liegen die Magnete direkt aufeinander, somit besteht die Gefahr, dass beim Trennen der Boxen die Magnete aus ihrer Halterung herausgerissen werden.

2. Eine andere Idee ist, 90 Grad Aluminiumwinkel zu benutzen. Hierbei ergibt sich der Vorteil, dass die Magnete von innen in das Aluminium versenkt werden können. Es bleibt von außen gesehen eine 1mm dicke Schicht Aluminium. Beim Aufbau des kompletten Feldes verhindert diese, dass zwei Magnete direkt aufeinander liegen. Die Seitenflächen werden bei dieser Lösung von innen festgeschraubt.

Die Entscheidung fiel auf die Aluminiumwinkel, da diese Lösung keine größeren Nachteile besitzt. In den nächsten beiden Kapiteln wird die Bearbeitung der Aluminiumwinkel und der Seitenflächen getrennt beschrieben.

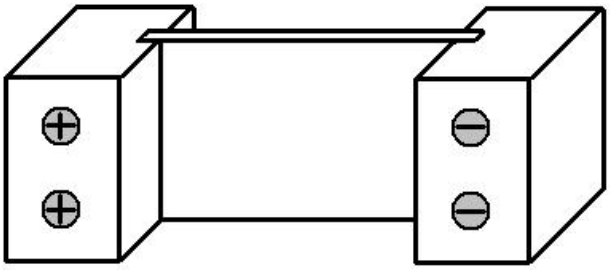


Abbildung 3.58: Ansicht der Unterkonstruktion mit Blöcken

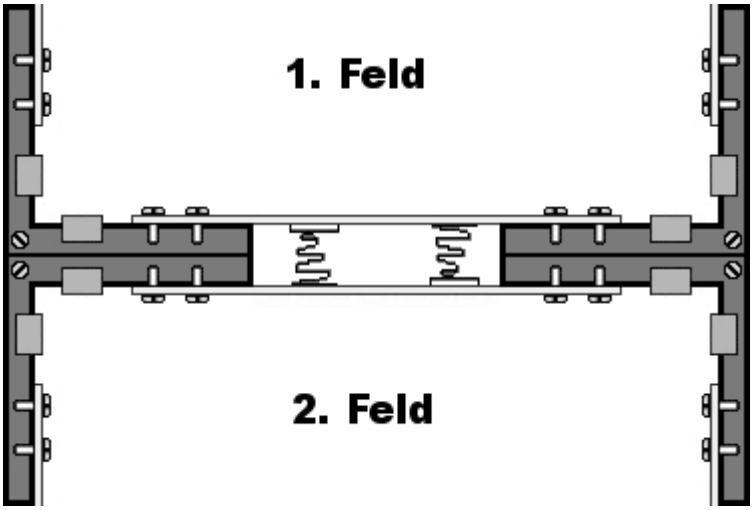


Abbildung 3.59: Zusammenhalten der Gesamtkonstruktion durch Magnete

3.4.3.1 Seitenflächen

Die Seitenflächen sind sowohl für die Stabilität als auch für die Befestigung der Kontakte zwischen den Boxen wichtig.

Zusätzlich zu den oben schon beschriebenen Materialien (Holz, Metall und Plastik) kommen hierfür noch Epoxydharzplatinen in Betracht. Ein wesentlicher Vorteil von Platinen gegenüber den anderen Materialien liegt darin, dass die Kontakte direkt auf die Platine gelötet werden können. Somit entfällt das Problem der Befestigung der Kontakte. Weitere Vorteile liegen darin, dass Platinen auch in einer sehr geringen Dicke sehr stabil sind und von allen Materialien am günstigsten zu beschaffen sind. Da Platinen so gut wie keine Nachteile haben, werden die Seitenflächen aus diesem Material hergestellt.

Damit die Platine sicher mit den Winkeln verschraubt werden kann, sollte sie wesentlich größer als 5cm sein. Wie in Kap 3.4.3.2 zu sehen, ist der freie Bereich zwischen den Winkeln exakt 5cm groß. Da in den Winkel zusätzlich noch Magnete untergebracht werden müssen, kann die Platine aber nicht größer als 10cm sein. Um einen optimalen Halt zu gewährleisten, ist für die Platinen diese maximale Breite gewählt worden.

Die Platine darf nicht höher als 4cm sein, da auf ihr die Led-Platine befestigt wird. Dementsprechend hat die Platine eine Größe von 4cm x 10cm.

3.4.3.2 Winkel

Die Winkel befinden sich dabei in den 4 Ecken des Gehäuses, wie in Abbildung 3.60 zu sehen. Sie bestehen aus Aluminium.

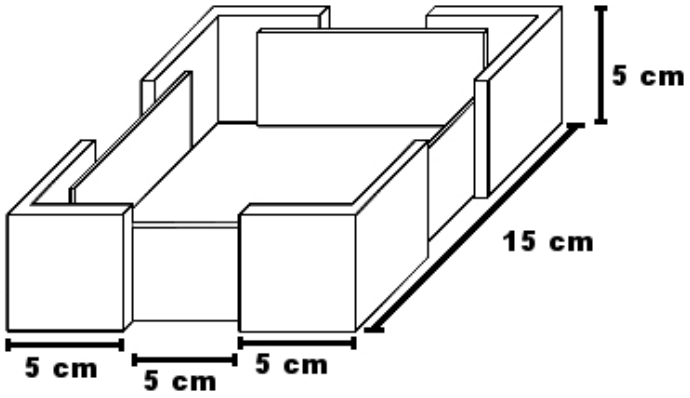


Abbildung 3.60: Ansicht der Unterkonstruktion

An den Winkeln werden die Seitenflächen, wie in Abbildung 3.61 zu sehen, befestigt. Dies geschieht durch jeweils 2 Gewinde pro Winkelfläche und sorgt für die benötigte Stabilität der Unterkonstruktion. Durch die in die Winkel eingelassenen Magnete wird sichergestellt, dass die Gesamtkonstruktion zusammenhält. Die Magnete befinden sich hierbei im Inneren des Winkels. Dies garantiert einen optimalen Halt zwischen den Spielfeldboxen.

Die Größe der Winkel beträgt 5cm x 5cm bei einer Dicke von 6mm. Die Höhe ergibt sich aus den innen liegenden Bauelementen (siehe Kapitel 3.4.6). Aufgrund des Durchmessers der Magnete und den neben liegenden Gewinden beträgt die Breite auch 5cm. Die Dicke von 6mm ergibt sich daraus, dass die Magnete in den Winkeln versenkt werden sollen.

3.4.4 Oberfläche

Über die Oberfläche der Spielfeldboxen müssen die Roboter fahren können. Die Oberfläche muss stabil genug sein, den Roboter tragen zu können. Außerdem muss die Ober-

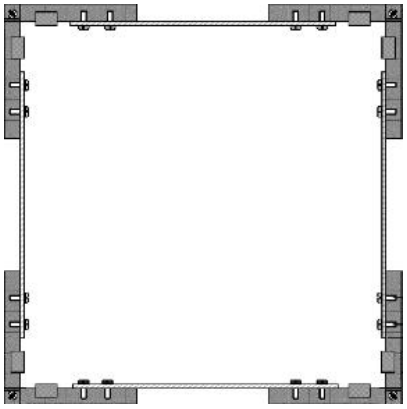


Abbildung 3.61: Aufsicht der Unterkonstruktion

fläche ein grafisches Design haben, so dass der Spieler erkennen kann, welches Feld welche Eigenschaften hat. Ferner sollte die Oberfläche natürlich länger haltbar sein. Außerdem sollte es möglich sein, Checkpoints festzulegen.

3.4.4.1 Grafisches Design

Das grafische Design der Oberfläche dient dazu, dass der Benutzer erkennen kann, welches Feld welche Eigenschaften hat (z. B. wo sich eine Wand befindet). Da die Roboter über die Felder fahren müssen, gibt es leider keinerlei Möglichkeiten, das Design 3-Dimensional zu gestalten. In Anlehnung an das englische Originalbrettspiel ist das Layout der einzelnen Felder der Spielbretter des Brettspiels verwendet worden und diese zu diesem Zweck eingescannt worden. Es sind einzelne Felder ausgesucht worden, die zu 3 x 3 Feldern zusammengesetzt worden sind. Diese 3x3 Felder sind auf 120g Papier gedruckt worden.

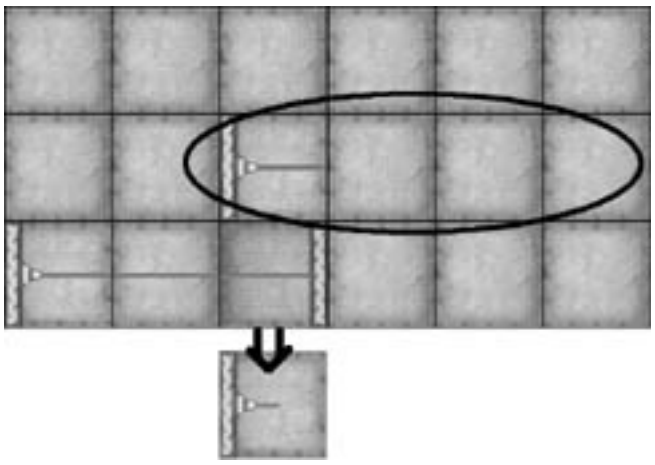


Abbildung 3.62: Spielfläche mit Laser

Allerdings gab es hierbei ein Problem mit der Darstellung der Laser. Wenn ein Laser auf einer Spielfeldbox beginnt, muss dieser nicht unbedingt auf derselben enden (siehe schwarzer Kreis in Abb. 3.62). Leider kann dieses nicht auf dem Papier dargestellt werden. Deswegen wird nur der Anfangspunkt des Lasers grafisch auf dem Papier festgehalten. Damit der Spieler sich nicht vorstellen muss, wo sich ein Laserstrahl auf dem Spielfeld befindet, werden dazu die in Kapitel 3.2.3 beschriebenen LEDs benutzt. In der Programmierphase werden an den Stellen wo sich Laserstrahlen auf dem Feld befinden, die jeweiligen LEDs eingeschaltet. Auf diese Weise sind die Laser nicht auf eine Länge von 3 Feldern beschränkt.

3.4.4.2 Haltbarkeit der Oberfläche

Zunächst wurde überlegt, dass das Papier auf eine Platte geklebt wird. Dabei ergab sich leider das Problem, dass einige Löcher für die bereits beschriebene Kommunikation gebohrt werden müssen. Ein weiteres Problem liegt darin, dass sich die Bedruckung auf Dauer während des Spiels abnutzen würde. Aus diesem Grund ist die Idee entstanden, dass das bedruckte Papier unter einer 1,5mm dicken 15x15cm großen Makrolon-Platte befestigt wird. Da das Makrolon durchsichtig ist, müssen keine Löcher für die Kommunikation in das Makrolon gebohrt werden. Um das Makrolon mit der Unterkonstruktion zu verbinden, haben sind in die Ecke der Aluminiumwinkel Gewinde geschnitten worden und die Makrolon-Platte ist mit Schrauben an den Winkeln befestigt worden. Damit der Roboter problemlos über diese Oberfläche fahren kann, sind zu diesem Zweck Senkkopfschrauben verwendet worden.

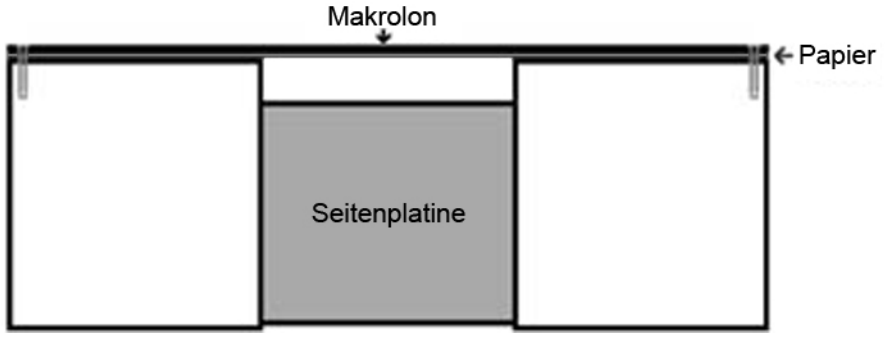


Abbildung 3.63: Befestigung der Makrolon-Platte

3.4.4.3 Checkpoints

Die Oberfläche der Spielfeldbox muss eine Möglichkeit bieten, die schon oben beschriebenen Checkpoints festlegen zu können, siehe [AMI99]. Dazu ist folgende Idee entstanden: In Anlehnung an das englische Spiel werden die Checkpoints als Fähnchen realisiert. Die Fähnchen werden in einer Ecke des Feldes platziert, und über Drucktaster auf der

Platine wird festgestellt, wo ein Fähnchen steckt. Da die Roboter 4,5cm groß sind und wahrscheinlich nicht exakt in der Mitte der Felder fahren, müssen die Fähnchen möglichst weit vom Mittelpunkt des Feldes entfernt sein. D.h. die Drucktaster für die Fähnchen können leider nur in den Ecken der Felder platziert werden. Da auf der Platine der Platz für die Drucktaster vorhanden sein muss, bleiben so nur noch 4 Plätze für Drucktaster (siehe Abb. 3.64).

Dementsprechend können sich auf jeder Spielfeldbox nur 4 Felder befinden, die ein Checkpoint sein können. Darauf muss beim grafischen Design natürlich geachtet werden. Dies ist vor allem deshalb möglich, da laut Spielbeschreibung nur folgende Felder ein Checkpoint sein können (siehe [AMI99]):

- Feld ohne eine besondere Funktion
- Feld mit nur einer Mauer
- Feld mit 2 Mauern
- Reparaturfeld ohne Mauer
- Reparaturfeld mit einer Mauer
- Doppeltes Reparaturfeld ohne Mauer
- Doppeltes Reparaturfeld mit einer Mauer

Es ist also ohne Probleme möglich alle Boxen so zu gestalten, dass höchstens 4 Felder auf jeder Box vorhanden sind, die ein Checkpoint sein können.

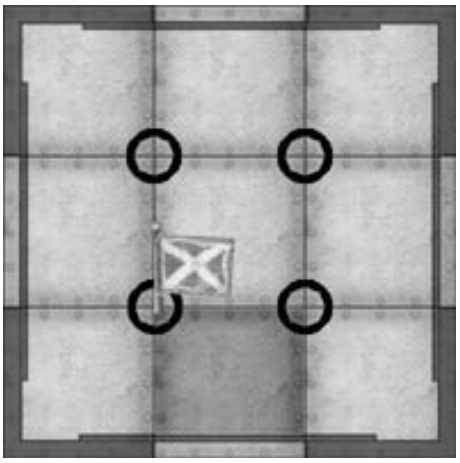


Abbildung 3.64: Plätze für die Drucktaster in der Spielfläche

Jeder Checkpoint hat eine eindeutige Nummer und diese muß festgestellt werden. Da durch die Drucktaster nur festgestellt werden kann, ob ein Feld ein Checkpoint ist oder nicht, muß die Nummer des Checkpoints durch die Software festgestellt werden. Der Host-PC wird den Spieler dazu auffordern den x-ten Checkpoint einzustecken. In dem Moment, wo ein Fähnchen in ein Feld eingesteckt wird, wird dies durch den Drucktaster festgestellt. Das Feld bekommt danach von dem Host die Nummer des Checkpoints, siehe Kap. 4.3.3.3.

3.4.5 Verbindungen der Felder

Es gibt drei Arten der Verbindungen: Zum Einen solche, die die Stabilität des Gesamtfeldes erzeugen und zum Anderen die Verbindungen, die zur Übertragung der Kommunikation oder des Stromes dienen. Die Seitenplatine bildet die Grundlage zur Übertragung der Kommunikation. Auf dieser Platine werden die Kontakte und die entsprechenden Gegenstücke montiert. Jede dieser Seitenplatinen ist mit einem negativ gepolten Winkel verbunden, der für diese Platine zusätzlich als Masse dient. Die Winkel werden zur Stromübertragung genutzt und halten durch die permanent Magneten das Gesamtfeld zusammen.

3.4.5.1 Zusammenhalten der Felder

Zwei Möglichkeiten der Realisierung von festen Verbindungen sind denkbar.

Die Eine ist eine Kombination von Steckern und Buchsen. Hier werden auf jeder Seite männliche wie weibliche Stecker angebracht. Durch die Anforderung der beliebigen Zusammensteckbarkeit kann diese Idee nicht realisiert werden, da hier zwei Seiten mit weiblichen Steckern nicht kombiniert werden können.

Daraus ergibt sich, dass nur Kontakte verwendet werden können, die entweder auf beiden Seiten gleich aussehen oder wie beim Prinzip der Batterie, Federn mit Metallplatten kombinieren.

Es bleiben Federbügelkontakte und Batteriefedern mit zugehörigen Gegenstücken.

Diese Art von Kontakten hat das Problem, dass die Boxen nicht durch die Kontakte selber zusammengehalten werden. Um trotzdem einen guten Halt zwischen den Feldern zu gewährleisten, werden Magnete in die Winkel versenkt (siehe Abbildung 3.59).

Hierbei müssen,wie in Abbildung 3.65 zu sehen, je zwei Magnete pro Winkelseite übereinander gesetzt werden, um den benötigten Druck zwischen den Boxen zu erzeugen. In die eine Seite des Winkels werden die Magnete mit der negativen Seite nach außen befestigt. Auf der anderen Seite geschieht dies entgegengesetzt. Dies garantiert die Drehbarkeit.

Wird nun eine komplette Spielfeldboxseite betrachtet, ergibt sich das in Abbildung 3.66 zu sehende Bild.

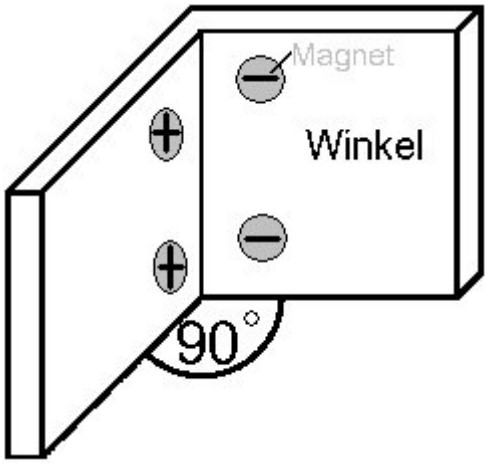


Abbildung 3.65: Polung der Magnete in einem Winkel

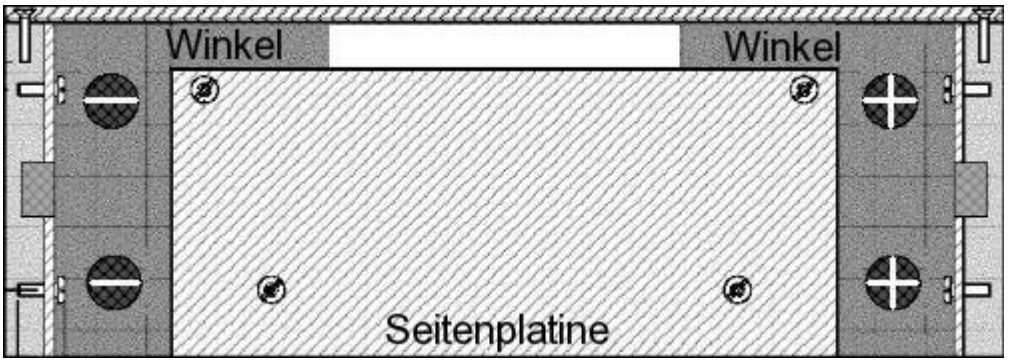


Abbildung 3.66: Polung der Magnete für eine Spielfeldseite

3.4.5.2 Kontakte zur Übertragung der Kommunikation

Für diese Kontakte können nur Federbügelkontakte und Batteriefedern verwendet werden, da die Realisierung von Steckern/ Buchsen wie oben beschrieben nicht möglich ist. Um die Drehbarkeit der Spielfeldboxen zu gewährleisten, werden zur Realisierung der Daten- und Takt- Leitungen, insgesamt 4 Kontakte benötigt.

Ein passender Federbügelkontakt ist der Kontakt von Elco mit einer Größe von 1cm x 0,8cm x 5,5mm; zu sehen in Abbildung 3.67. Er besitzt 4 Kontakte je Einheit mit einem Abstand von 1,1mm. Auf jeder der Seitenplatinen wird ein Elco-Kontakt angelötet.

Hier ergibt sich das Problem, dass es bei geringer Verschiebung der gegenüberliegenden Kontakte zu ein Kurzschluss kommen kann. Zusätzlich haben sich die Federbügel des Kontaktes als nicht widerstandsfähig genug erwiesen.

Abbildung 3.67: Federbügelkontakt von Elco

Weitere für uns passende Kontakte sind die Batteriefedern von Conrad. Diese sehr stabilen Federn haben eine Höhe von 1,1cm. Um diese Federn zu verwenden zu können, werden auf Grund der Anforderung der Drehbarkeit der Spielfelder pro Seitenfläche 2 Federn angelötet. Eine Feder dient als Daten-Leitung und die andere als Takt-Leitung. Um den Kontakt zum anderen Spielfeld herzustellen, muss eine Unterlegscheibe als Gegenstück verwendet werden, da die Winkel eine Dicke von 6mm haben und sich insgesamt also ein Abstand von 1,2 cm ergibt. Dies ergibt folgende Seitenansicht der Box (siehe Abbildung 3.68).

Weitere Anforderungen sind die Lötbarkeit des Materials und die Lochgröße der Unterlegscheibe. Bei der Lochgröße ist zu beachten, dass sie den Durchmesser der Feder unterschreiten muss.

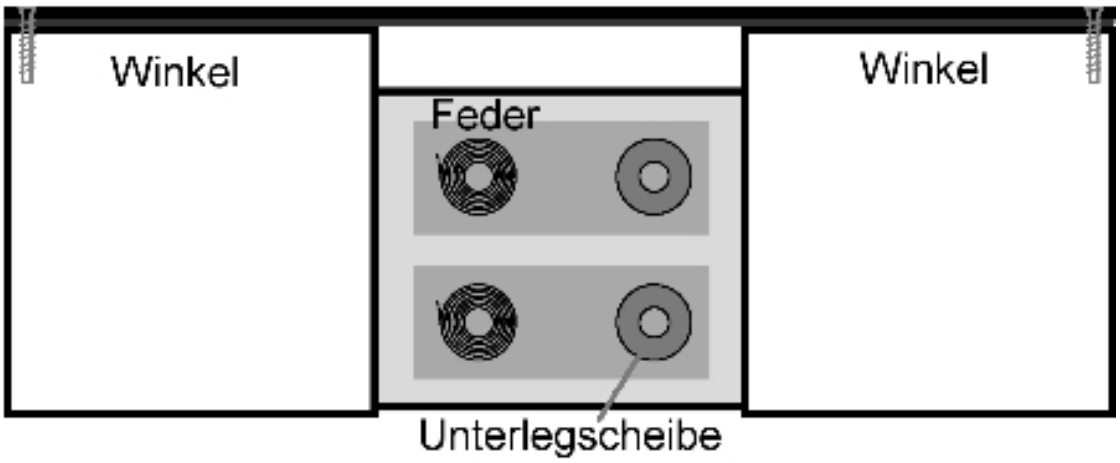


Abbildung 3.68: Seitenansicht auf eine Spielfeldbox

3.4.5.3 Stromversorgung

Die Stromversorgung wird über den Parkplatz geregelt, da alle anderen Spielfeldboxen frei drehbar sein müssen und somit keinen geeigneten Stromanschluss darstellen.

Hier wird der Strom über die Winkel verteilt. Um die Drehbarkeit der Spielfeldboxen zu garantieren, müssen die Winkel mit gleichem Potential diagonal angeordnet und über die Platine verbunden werden. Dies ergibt folgendes Bild (siehe Abbildung 3.69).

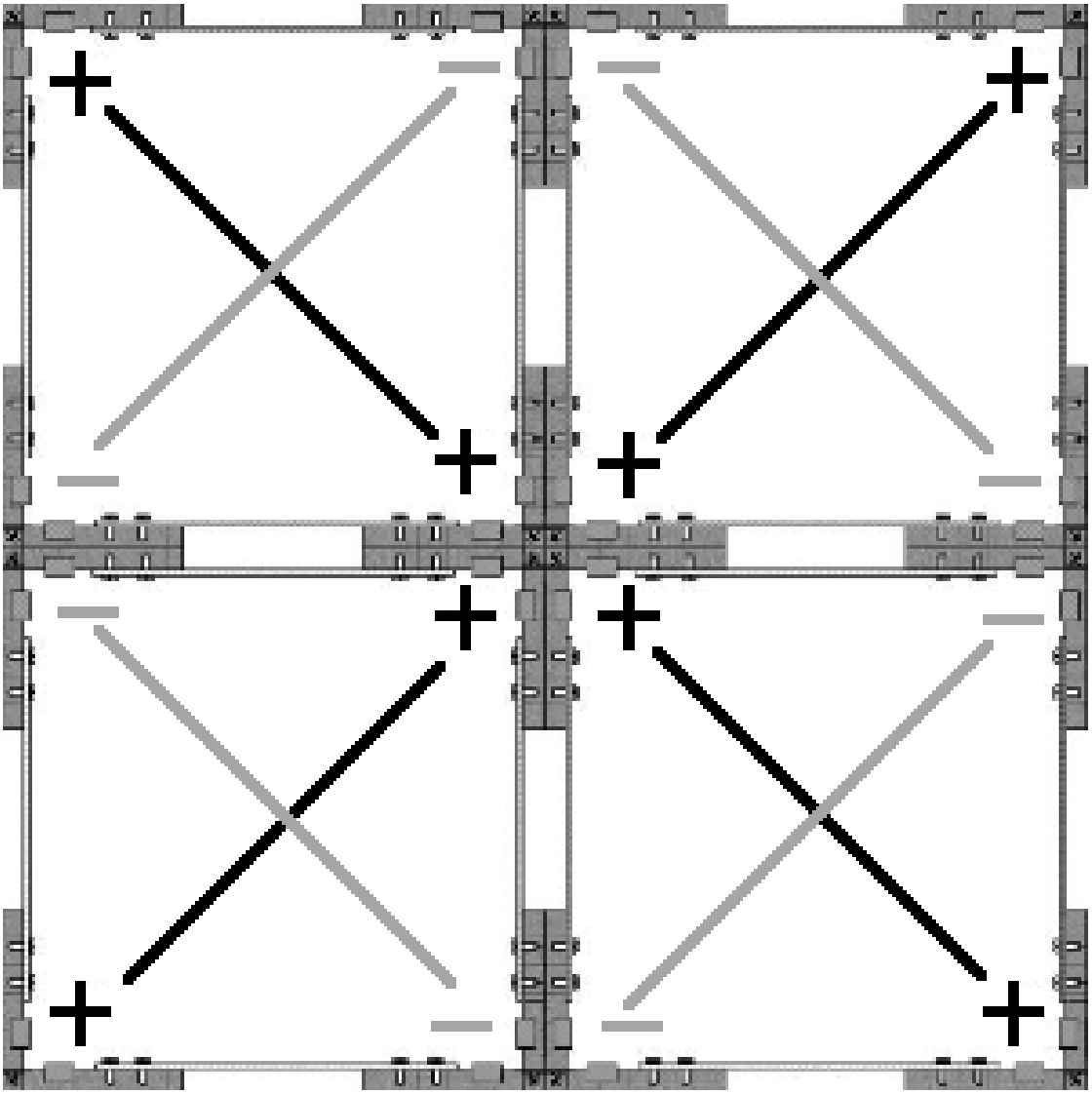


Abbildung 3.69: Stromversorgung über die Winkel

3.4.6 Elektronischer Aufbau der Spielfeldes

Zu Beginn der PG musste geklärt werden welche Anforderungen die Spielfeldboxen zu erfüllen haben. Folgende Punkte wurden herausgestellt:

- Jedes Feld soll 4 LEDs zur Navigation bekommen. Diese LEDs sollen mit gleichem Abstand zum Mittelpunkt jeweils in allen vier Himmelsrichtungen platziert werden. Näheres siehe Kapitel 3.4.7.1.
- In der Mitte wird eine Leuchtdiode platziert, die sowohl für die Navigation als auch für die Kommunikation zum Einsatz kommen soll.
- Neben der Kommunikations-LED (mittlere Leuchtdiode) sollen jeweils Fototransistoren platziert werden, mit denen festgestellt werden soll, ob sich ein Roboter auf dem Feld befindet. Die Platzierung der Leuchtdioden und Fototransistoren ist Abbildung 3.72 zu entnehmen.
- Weiterhin muss eine Markierung einzelner Felder als Checkpoints möglich sein.
- Zur Kommunikation mit anderen Spielfeldern werden Anschlüsse an jeder Seite der Spielfeldbox benötigt. Zu Beginn wurden dafür 4 Pins pro Seite eingeplant. Im weiteren Verlauf stellte sich jedoch heraus, dass 2 Pins hier ausreichen.
- Die Spielfelder sollen Geräusche zu bestimmten Aktionen, die auf dem Spielfeld stattfinden, abgeben.

Daraus ergibt sich ein Bedarf von 63 Anschlüssen zur Ansteuerung der LEDs, Phototransistoren und Checkpoints. Zusätzlich werden noch acht Anschlüsse für die Kommunikation und 1-2 Anschlüsse für den Sound benötigt. Die Suche nach Mikrocontrollern mit entsprechend vielen Ausgängen verlief wenig erfolgreich. Zum Ersten bieten nur wenige Hersteller solche Mikrocontroller überhaupt an. Zum Zweiten sind diese speziellen Controller dann nicht oder nur in großer Stückzahl lieferbar.

Als Ausweg wurde gewählt, die reinen Navigations-LEDs über eine gesonderte Schaltung mit TTL-Bausteinen anzusteuern. Dies kann die Schaltgeschwindigkeit der LEDs beeinflussen. Da die Navigations-LEDs aber nicht allzu oft geschaltet werden, führt dies auch nicht zu Einschränkungen im Spielbetrieb. Dadurch werden die benötigten Anschlüsse für die Navigation allerdings von 36 auf 6-8 Anschlüsse (4-6 für die Adressierung, 1 Data, 1 Clock) reduziert. Die Kommunikations-LEDs in der Mitte der Felder sowie die Fototransistoren sollen direkt vom Mikrocontroller angesteuert werden, um eine reibungslose Kommunikation zu gewährleisten. Damit bleibt die Anzahl der benötigten Anschlüsse hier bei 16. Für die Checkpoints ist nur ein zusätzlicher Anschluss nötig, da die einzelnen Checkpoints über die Adresse der Navigations-LEDs angesteuert werden können. Die Kommunikation benötigt nur noch 2 Anschlüsse pro Spielfeldseite, also insgesamt 8. Für den Sound ist ebenfalls ein weiterer Anschluss nötig. Insgesamt werden also 36 Anschlüsse benötigt.

Als Mikrocontroller galt von Beginn an der LPC2114 von Phillips als aussichtreichster Kandidat für den Einsatz im Feld. Für die Ansteuerung der Navigations-LED und die Abfrage der Checkpoint-Taster wurde neben der Verwendung von den oben angesprochenen TTL-Bausteinen noch eine weitere Lösung gefunden. Der Einsatz eines CPLD (Complex Programmable Logic Device) ermöglicht die Realisierung der gleichen Funktionen in einer One-Chip Lösung und wurde als Alternative parallel entwickelt werden.

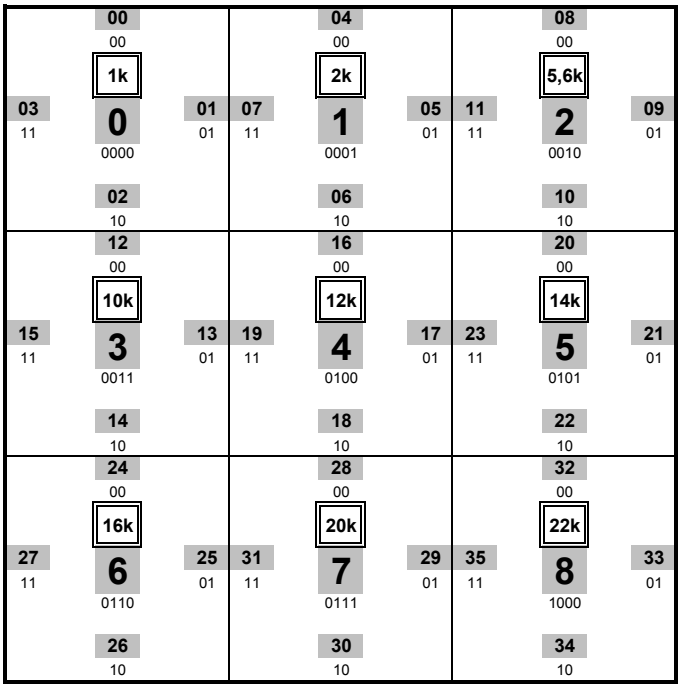


Abbildung 3.70: Adressen der Felder und LEDs

Zu Beginn wurde außerdem die in Abbildung 3.70 dargestellte Adressierung der LEDs festgelegt. Demnach werden die einzelnen Felder jeder Spielfeldbox mit 0 (beginnend oben links) bis 8 (endend unten rechts) durchnummeriert. Damit werden vier Bits zur Adressierung benötigt. Die Navigations-LEDs auf jedem Feld werden im Uhrzeigersinn, beginnend mit der obersten LED, ebenfalls von 0 bis 3 durchnummeriert. Die zwei hierfür benötigten Bits werden an die vier Bits für die Adressierung der Felder angehängt. Somit ist jede LED über eine 6 Bit Adresse ansteuerbar. Für die Ansteuerung mit TTL-Bausteinen ergab sich allerdings eine andere Adressierung. Hierzu siehe Kapitel 3.4.7.2.

3.4.7 Testboards

Aus den obigen Annahmen ergaben sich die zu fertigenden Komponenten für das Feld. Es wurde beschlossen ein Feld zu entwickeln, das in Einzelteilen auch getestet werden kann. Dazu müssen die verschiedenen Bauteilgruppen auf einzelne Platinen aufgeteilt werden, so dass ein unabhängiges Testen ermöglicht wird. Die Platinen sollen weiterhin die Möglichkeit besitzen, direkt miteinander verbunden zu werden, so dass ein komplettes Spielfeld zusammengebaut werden kann. Dabei ergab sich eine Unterteilung in folgende fünf Komponenten:

- **Feld:** Eine Platine soll die LEDs des Spielfeldes, die Fototransistoren mit Verstärkerschaltung und die Taster für die Checkpoints enthalten. Eine Erläuterung der Feldplatine findet in Kapitel 3.4.7.1 statt.
- **TTL-Bausteine / CPLD:** Die zweite Platine übernimmt dann die Ansteuerung der Navigations-LEDs und Checkpoints. Sie entsteht in zwei Versionen. Zum Ersten eine Version mit TTL-Bausteinen und zum Zweiten eine Version mit einem CPLD. In Kapitel 3.4.7.2 wird die Schaltung mit TTL Bausteinen und in Kapitel 3.4.7.3 die Schaltung mit dem CPLD erläutert.
- **Mikrocontroller:** Die dritte Platine beherbergt den Mikrocontroller und alle dazugehörigen Komponenten.
- **Spannungsversorgung:** Im Spielfeld werden konstante Spannungen von 3,3V und 1,8V benötigt. Da am Spielfeld eine höhere Spannung anliegt, ist eine extra Schaltung nötig. Erläutert wird diese in Kapitel 3.4.7.4.
- **Sound:** Zur Erzeugung von Geräuschen wird der PWM-Ausgang des Mikrocontrollers benutzt. Dazu wird eine kleine Verstärkerschaltung benötigt. In Kapitel 3.4.7.5 wird diese beschrieben.

Die Verbindung der Platinen erfolgt über Stecker und Buchsenleisten zwischen Feld- und TTL-/CPLD-Platine und einem Flachbandkabel zwischen TTL-/CPLD- und Mikrocontrollerplatine. Die TTL-/CPLD-Platine wird dabei direkt unter die erste Platine gesteckt, und beide Platinen zusammen dann ins Spielfeld montiert. Die dritte Platine mit dem Mikrocontroller muss außerhalb der Spielfeldbox platziert werden und wird mit einem 50 poligen Flachbandkabel verbunden. Die Anschlussbelegungen und die Position der Stecker sind Abbildung 3.71 dargestellt.

3.4.7.1 Feldplatine

Die Feldplatine soll die Navigations- und Kommunikations-LEDs, Taster für die Checkpoints und die Fototransistoren incl. Verstärkerschaltung enthalten. Weiterhin sollen zwei 40-polige Buchsen die Schnittstelle zu der TTL-/CPLD-Platine bilden.

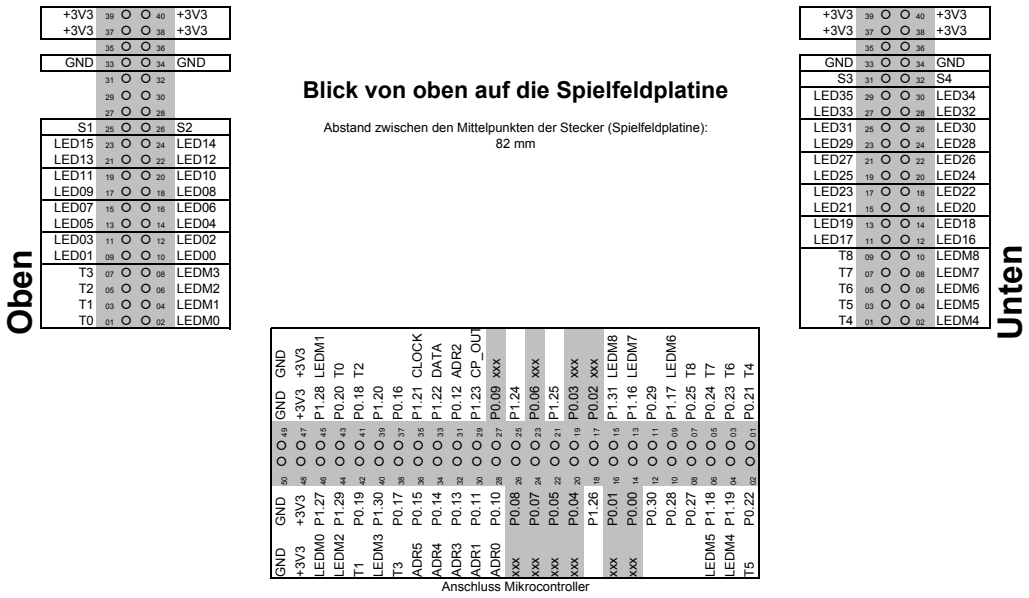


Abbildung 3.71: Steckerbelegung der Testboards für das Spielfeld

Die Kommunikations-LEDs befinden sich dabei in der Mitte jedes Feldes. Wie oben schon erwähnt, werden die Navigations-LEDs jeweils mit einem Abstand von 16,66 mm in alle vier Himmelsrichtungen um die jeweilige Kommunikations-LED eines Feldes platziert. Der Abstand aller Leuchtdioden zueinander ist damit gleich, so dass auch gleichmäßige Linien abgebildet werden können, die die Laserstrahlen simulieren sollen. Die Position der jeweiligen Navigations-LEDs ist damit außerdem noch soweit vom Mittelpunkt entfernt, dass eine gute Ausrichtung des Roboters möglich ist. Eine weitere Verschiebung nach außen wäre auch nur noch in geringem Umfang möglich, da die LEDs sonst nicht mehr unter dem mittig auf dem Feld stehenden Roboter liegen würden. Die genaue Position der LEDs ist Abbildung 3.72 zu entnehmen. Der entwickelte Schaltplan befindet sich in Anhang A (Abbildung 6.4).

Um die Reaktion der Fototransistoren bei verschiedenen Lichtverhältnissen festzustellen wurden unterschiedliche Widerstände an den Fototransistoren eingelötet. Variiert wurde dabei die Widerstände RT0 bis RT8. Die Werte der Widerstände sind in Abbildung 3.70 festgehalten. Während der Tests wurde außerdem noch ein zusätzlicher 4,7 kOhm Widerstand zwischen dem Emitter des Fototransistors und Masse geschaltet.

3.4.7.2 LED-Ansteuerung mit TTL-Bausteinen

Für die Steuerung der LEDs, der Phototransitoren und der Taster verfügt der Mikrocontroller nicht über genug GPI/O-Pins. Hierbei ist die Idee entstanden eine extra Schaltung zu entwerfen, sodass wir mit weniger Pins auskommen.

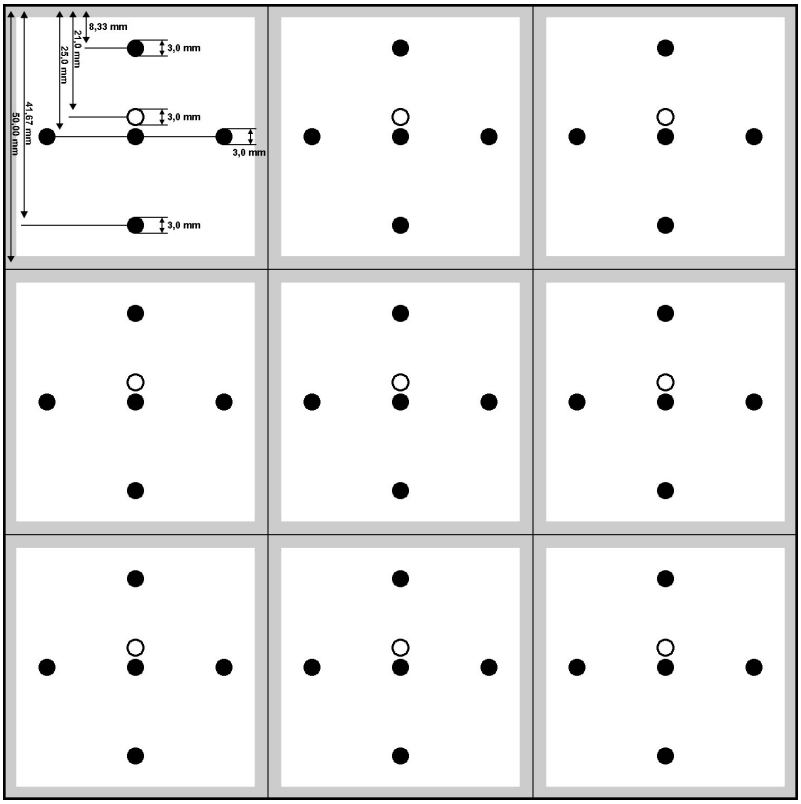


Abbildung 3.72: Position der Dioden und Phototransistoren

- 5 8-Bits adressierbare Latch (74HCT259),
- 1 8-Bits adressierbare Latch (74HCT138),
- 1 8-Input Multiplexer (74AC151)

Die einzelnen Bauteile werden in den nächsten Kapiteln erläutert.

Adressierbare Latches

74HCT259 Die adressierbaren Latches sind Halbleiterbauteile, die aus wenigen Eingangssignalen mehrere Ausgangsignale liefern können. Die Latches verfügen über Speicher, in denen die Ausgabewerte gespeichert werden. Für die Schaltung sollten 16-Bits adressierbare Latches verwenden werden, um wenige Bauteile auf der Platine zu haben. Da diese nicht mehr verfügbar waren, wurde eine alternative Entscheidung herangezogen, so dass nun 8-Bits adressierbare Latches verwenden wurden, die ausserdem noch günstiger sind. Die 8-Bits adressierbare Latch “74HCT259” von dem Hersteller Philips waren für die zu realisierenden Aufgabe geeignet. Diese Bausteine bieten den Vorteil, dass jede Ausgangsleitung über einen eigenen Speicher verfügt. Der 74HCT259 besteht aus 16 Pins

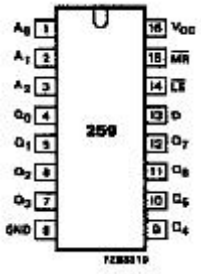


Abbildung 3.73: 74HCT259-Baustein

wobei drei (A0 ,A1, A2) für die Adressierung verwendet werden, die weiteren sieben Pins (Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7)sind Ausgangspins. Der Pin “MR” wird verwendet, um den Inhalt des Speichers zu löschen. “LE” kontrolliert den Zustand des Latches. “GND” soll mit der Masse-Leitung verbunden werden, während “VCC” für die Stromversorgung zuständig ist. “D” wird verwendet, um die Ausgabedaten in den Speicher zu legen. Nachdem der Latch angeschlossen ist, soll “LE” auf Masse gezogen werden. Dieser Zustand ermöglicht die Speicherung der Ausgabewerte in den jeweiligen Speicher des Latches. Die Adresse wird durch das entsprechende Setzen der Pins A0,A1,A2 zwischen 000 bis 111 gewählt. Ein LOW auf dem Pin “LE” setzt den Latch in den Lesezustand. Dies wird ausgeführt, indem das Signal, das im Pin D anliegt, in der ausgewählten Ausgangsadresse gespeichert wird. Nach der Speicherung des Ausgangssignals in den jeweiligen Speichern des Latches wird der Pin “LE” auf High gesetzt. Dieser Zustand entspricht der Ausgabe des jeweils gespeicherten Signals. Die LEDs werden invertiert geschaltet, da der Mikrocontroller nicht genug Strom liefern kann, um alle LEDs einzuschalten. Die Ausgangspins des Latches sind dann auf Masse gezogen (auf LOW gesetzt), wenn die LED blinken soll. (siehe Abbildung 3.4.7.1)

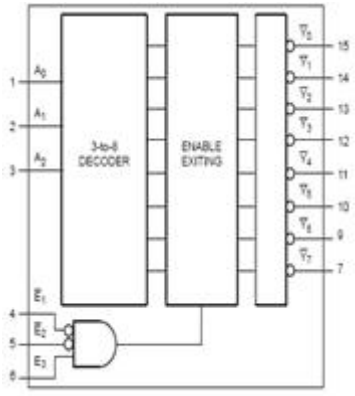


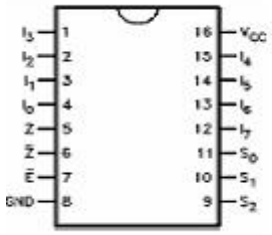
Abbildung 3.74: 74HCT138-Baustein

74HCT138 Für eine weitere Optimierung haben wir ein weiteres 8-Bits adressierbares Latch vom Typ "74HCT138" verwendet. Dieses Bauteil hat im Vergleich zum "74HCT259" den Vorteil , dass das Ausgangssignal invertiert wird.

"74HCT138" enthält genauso wie "74HCT259" 16 Pins, wobei 3 Pins für die Adressierung, ein Pin für den Strom und ein weiterer Pin für die Masse verwendet werden. Die drei Pins E1, E2, E3 werden verwendet, um den Zustand des Bausteins zu definieren. E1 und E2 werden auf Masse gezogen, während E3 auf High gesetzt wird. Wenn eine Adresse gewählt wird, wird der Ausgabewert auf diesem Pin LOW, während alle anderen Pins HIGH ausgeben. Der "74HCT138" enthält keinen Pin für die Datenleitung. Nur der Ausgabewert auf der ausgewählten Adresse ist LOW gesetzt.

Multiplexer

Für die Steuerung der Checkpoints haben wir einen Multiplexer (Abb. 3.64) verwendet. Dieser Baustein besteht aus 8 Pins (I0,...,I7) für die Signaleingabe, zwei Pins (Z,Z) für die Signalausgabe und drei Pins (S0, S1, S2) für die Adressierung des Eingangspins. Der Multiplexer erhält zu jedem Zeitpunkt 8 verschiedene Signale. Nur eines der 8 Eingabesignale wird ausgegeben, indem die Adresse gewählt wird. Der Mikrocontroller soll nur überprüfen, ob der Drucktaster gedrückt ist, der den Checkpoint charakterisiert (das heißt, es wird überprüft ob Strom fließt). Falls Strom durch diese Leitung fließt, interpretiert der Mikrocontroller dieses Feld als ein Checkpoint.



Pin Names	Description
I_0–I_7	Data Inputs
S_0–S_2	Select Inputs
$\overline{E}$	Enable Input
Z	Data Output
$\overline{Z}$	Inverted Data Output

Abbildung 3.75: Multiplexer

Schaltung

Die Schaltung wurde mit dem Programm Eagle entworfen (Abb. 3.4.7.2). Die LEDs und Fototransistoren, die für die Kommunikation mit dem Roboter vorgesehen sind, sind direkt mit dem Mikrocontroller verbunden. Für die Verbindung der Schaltung mit dem Mikrocontroller und der LED-Platine wurden drei Steckverbinder verwendet. Eine wird mit dem Mikrocontroller angeschlossen und die zwei anderen mit der LED-Platine. Für die Steuerung der LEDs, die für die Orientierung der Roboter verwendet werden, werden sieben GPI/O Pins des Mikrocontrollers verwendet. Drei davon sind für die Adressierung

der LEDs und der Drucktaster, einer für die Datenleitung und drei für die Adressierung des Latches (74HCT138), der mit den Enable-Pin auf den anderen Latches und der Multiplexer verbunden ist. 18 Pins werden als Kommunikationspins verwendet. Die Kommunikationsleitungen verbinden den Mikrocontroller direkt mit den mittleren LEDs und Fototransistoren auf der LED-Platine. In unserer Schaltung wird der “74HCT138” einzeln adressiert und mit den Enable-Pins alle anderen Bauteile verbunden. Dieser Baustein hat die Aufgabe zu entscheiden, welches Bauteil zu einem bestimmten Zeitpunkt aktiv sein darf, indem er den Enable-Pin des gewählten Bauteils auf LOW setzt. Dieser Vorgang ermöglicht, dass jede LED einzeln gesteuert werden kann. Die Eingabesignale, die über die Datenleitung von dem Mikrocontroller kommen, fließen in alle 5 Bauteile. Die 3 Pins der Adressierung sind mit den fünf Latches (74HCT259) verbunden. Über die Enable-Pins wird gewählt, welches Bauteil zu einem bestimmten Zeitpunkt das Signal auf der Datenleitung annehmen soll. Falls der Multiplexer aktiv ist, läuft der Mikrocontroller alle Eingabeadresse durch und überprüft, welche von ihnen Strom liefert. Die Steckleisten, die für die Verbindung der Steuerungsplatine mit der LED-Platine verwendet werden, werden auch für die Befestigung der beiden Platine benutzt. Da der Mikrocontroller nicht genug Strom für die Versorgung der LEDs und der Bauteile auf der Steuerungsplatine liefern kann, haben wir eine Versorgungsspannung auf der Steuerungsplatine vorgesehen. Diese Spannung sollte zwischen 3,3V und 5V sein. Die LEDs und die Bauteile werden direkt versorgt. Die shaltung wird im folgenden Bild dargestellt.

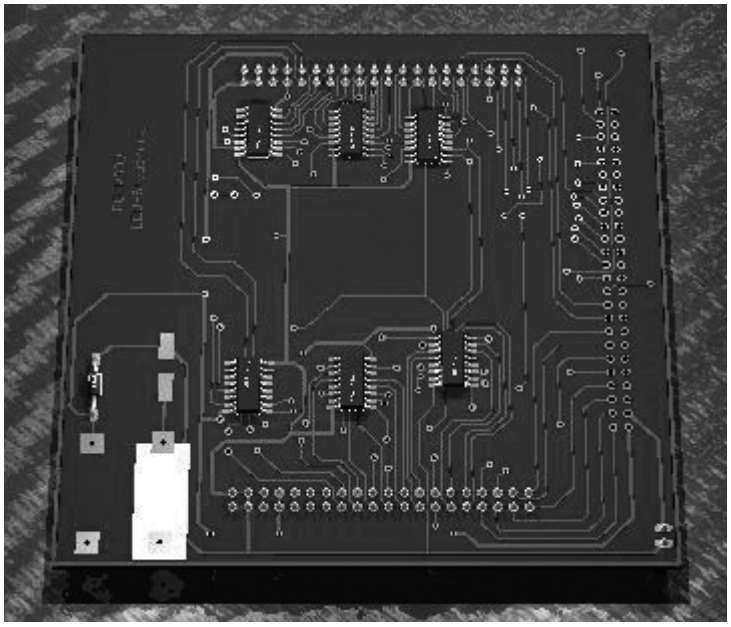


Abbildung 3.76: TTL-Platine

3.4.7.3 LED-Ansteuerung mit CPLD

Als zweite Möglichkeit, neben der im vorherigen Kapitel beschriebenen, soll eine Ansteuerung der LEDs durch einen CPLD entwickelt werden. Die CPLD-Variante ist in sofern von Vorteil, da sie sowohl Platz- als auch Preisvorteile bietet.

Hardware

Es stehen viele verschiedene Modelle diverser Hersteller zur Auswahl. Für die vorliegenden Gegebenheiten eignen sich besonders zwei Serien: Die MAX 3000A Familie von ALTERA und die XC9500 Serie von XILINX [Xil03]. Bei beiden Familien gibt es CPLDs, die alle Anforderungen erfüllen und auch nicht überdimensioniert sind. Aus Verfügbarkeits- und Preisgründen wurde ein Chip der XC9500 Serie von XILINX gewählt. Die genaue Bezeichnung lautet XC 9572XL TQ100. Dieser Chip hat 100 Pins, wobei 72 als I/O Pins eingesetzt werden können. Damit ist er mehr als ausreichend für die Anschlüsse der 36 Navigations-LEDs und der 4 Checkpoints, sowie der 9 Pins für die Ansteuerung und zum Auslesen der Checkpoints. Er kann außerdem mit einer Spannung von 3,3V betrieben werden, so dass er zum Mikrokontroller kompatibel ist. Weitere Informationen sind unter [Xil05c] zu finden.

Informationen zum Entwurf der Schaltung des CPLD sind unter [Xil99a], [Xil98b] und [Xil05a] zu finden. Eine spezielle Application Note beschreibt zudem den Anschluss von LEDs [Xil05b]. Mit diesen Angaben lässt sich leicht eine Schaltung erstellen. Die Platine selbst enthält dabei nur den CPLD mit einigen Entstörkondensatoren, zwei 40-polige Stecker für die Verbindung zur Feldplatine und eine 50-poliger Stecker zur Verbindung mit der Mikrocontrollerplatine, sowie einen Stromanschluss (siehe Schaltplan in Abbildung 6.5). Die Anschlüsse der Feldplatine, die direkt am Mikrocontroller angeschlossen werden, sind dabei einfach durchgeschleift. Eine Belgung der einzelnen Stecker ist Abbildung 3.71 zu entnehmen.

Software

Der CPLD benötigt allerdings auch noch eine Programmierung, damit die Ausgänge korrekt angesteuert werden können. Programme für den CPLD können mit der von Xilinx zur Verfügung gestellten Software, dem Xilinx WebPack in der aktuellen Version 7.1, erstellt werden. Die Programmierung erfolgt hier in VHDL. Probleme bei der Erstellung der Software entstehen durch einige Besonderheiten der Umsetzung des VHDL-Codes, so dass Programme dann zu groß für den CPLD werden können. Insbesondere die Verwendung von IF-Abfragen und Schleifen, bzw. eine Kombination davon ist sehr speicherintensiv. Eine Umstellung auf CASE-Anweisungen bringt eine Reduktion des Speicherbedarfs um mehr als die Hälfte, so dass nun die Größe des Programms kein Problem mehr darstellt.

Programmierung

Zur Programmierung des CPLD wird die dafür vorgesehene JTAG Programmierschnittstelle benutzt. Um das Programm mit der aktuellen Software (WebPack 7.1) auf den CPLD zu schreiben, muss ein Programmierkabel verwendet werden. Es stehen drei Alternativen zur Verfügung:

- Der Erwerb eines solchen Programmierkabels.
- Der Eigenbau eines Stecker nach einem von Xilinx [Xil98a] bereitgestellten Schaltplan.
- Die Verwendung eines älteren am Lehrstuhl vorhandenen (XChecker-) Programmierkabels, das allerdings nicht mit der aktuellen Software-Version kompatibel ist.

Die erste Variante scheidet bei Preisen von weit über 100 Euro für solche Kabel aus. Der Eigenbau der Schaltung für das Programmierkabel ist ebenfalls mit relativ hohem Aufwand und auch einigen Kosten verbunden, so dass die dritte Möglichkeit die Einfachste darstellt. Dazu muss allerdings zusätzlich das Programm JTAG-Programmer (Version 7.1), welches in alten Versionen des WebPacks (bis 4.3) enthalten ist, installiert werden. Ein Programmieren des CPLD direkt mit der neuen Software ist damit nicht mehr möglich.

Die Programme können nun mit dem ISE Webpack geschrieben und compiliert werden und anschließend mit dem JTAG Programmer auf den CPLD geschrieben werden. Informationen zur Programmierung sind unter [Xil02b], [Xil02a] und [Xil99b] zu finden. Besondere Einstellungen müssen nicht vorgenommen werden.

3.4.7.4 Spannungsversorgung

Die Spannungsversorgung einer Spielfeldbox geschieht über die Winkel (siehe Abschnitt 3.4.3.2) im mechanischen Aufbau. An diesen soll eine Gleichspannung anliegen. Da die Spielfläche variabel aufbaubar sein sollte, sind jeweils zwei diagonal gegenüber liegende Winkel elektrisch miteinander verbunden. Die zwei resultierenden Potentiale liegen wiederum an einem Brückengleichrichter an. Somit können die Potentiale durch das Drehen der Spielfeldbox vertauscht werden, und der Brückengleichrichter liefert eine richtig gepolte Spannung für den Spannungsregler.

Aufgrund der Dioden im Brückengleichrichter ist aber das Massepotential hinter dem Brückengleichrichter ein anderes als das der eingespeisten Gleichspannung. Daraus folgend wurde getestet, ob die Spielfeldboxen untereinander zu einer funktionierenden Kommunikation in der Lage sind, und ob die Verschiebung des Massepotentials keine Probleme verursachen wird.

Also wurde an einen im Hardwarelabor vorhandenen Brückengleichrichter eine Gleichspannung angelegt. Die Ausgangsspannung wurde an folgenden Spannungsregleraufbau

gelegt. Der Mikrocontroller benötigt zwei Spannungen, 1,8 V und 3,3 V. Aus Mangel einer günstigen und verfügbaren hybriden Lösung, wurden im Test die Spannungsregler LM1086CS-3.3 [Nat05] und XC6201P182MR [Tora] verwendet. Diese liefern auch im Host-Interface (siehe Abschnitt 3.6.2) die benötigten Spannungen. In der Spielfeldbox ist die SMD-Variante des LM1086 durch eine stehende Variante im TO220 Gehäuse für eine bessere Kühlung ersetzt. Außerdem wird der günstigere und leistungfähigere XC6206P182MR [Torb] verwendet.
Um den Eingangsspannungsbereich nicht durch den 1,8 V Spannungsregler zu beschränken, wird dieser durch den 3,3 V Spannungsregler gespeist. Der Aufbau enthielt weiterhin die in den Datenblättern empfohlenen Kondensatoren.

Mit den erzeugten Spannungen wurde ein Mikrocontroller-Testboard gespeist, welches durch Kommunikationsleitungen mit einem zweiten Mikrocontroller-Testboard verbunden war.
Das zweite Board wurde durch einen identischen Aufbau mit den benötigten Spannungen versorgt. Es gab also zwei Verbindungen der beiden Testboards, die im Prinzip zwei Spielfeldboxen darstellten. Die Erste war die in 3.2.2 beschriebene Kommunikationsverbindung, die auch zwischen den Spielfeldboxen so bestehen soll, und die Zweite war die Leitung für die an den Gleichrichtern anliegende Gleichspannung. Diese Verbindung entsteht im fertigen Aufbau der Spielfeldboxen durch die Winkel.
Den Abschluss des Tests bildete die erfolgreiche Kommunikation der Testboards untereinander. Dabei wurden Daten übertragen, die mittels der LEDs auf den Testboards dagestellt wurden.

3.4.7.5 Sound

Der Ablauf des Spiels soll mit Sound unterlegt werden. Dazu ist es notwendig, einen Lautsprecher so anzusteuern, dass eine ensprechende Klangwiedergabe möglich ist. Ein Lautsprecher besteht im Prinzip nur aus einer Spule und einer Membran. Je nach Stärke des Spulenmagnetfeldes bewegt sich die Membran entsprechend.

Da der Mikrocontroller bereits viele Anschlusspins belegt hat, ist es notwendig, die benötigte Anzahl von Pins zu minimieren. Dazu lassen sich die im Controller eingebaute PWM-Signalerzeugung nutzen. Ein PWM-Signal besteht lediglich aus einem Pin, der zu einem Zeitpunkt entweder im logischen Zustand Null oder Eins sein kann. Dabei wird der Wert über das Verhältnis zwischen der Anzahl der Einsen und der Anzahl der Nullen erzeugt.

Aufgrund des sehr begrenzten Speicherplatzes auf dem Controller können die Sounddaten nicht zu viel Platz in Anspruch nehmen. Eine angemessene Soundqualität lässt sich mit einer Samplerate von 8kHz mit einer Tiefe von 8 Bit erreichen. Diese Sounddaten werden als Konstante im ROM des Controllers gespeichert. Dazu benötigen wir pro Sample ein Byte. 8000 Samples pro Sekunde ergeben eine Datenmenge von 8000 Bytes/Sek.

Der PWM-Generator lässt sich durch folgende Werte einstellen.

- Taktfrequenz f - Durch den Controllertakt vorgegeben.
- Takte / Zyklus l - beliebig einstellbar, je nach Auflösung
- Dargestellter Wert d - beliebig wählbar zwischen 0 und l

Um den entsprechenden Wert darzustellen, muss der Zyklus durchgehend wiederholt werden. Dabei schaltet der Leitungszustand automatisch auf logisch Eins (= 3,3 Volt). Der Wert d gibt an, nach wie vielen Takten der Zustand auf Null schaltet (= 0 Volt). Nach l Takten wiederholt sich dieses Schema. Dabei stellt der Wert d ein einzelnes Datensample dar, welcher pro Sekunde 8000 mal geändert werden muss, der Sounddatei entsprechend. Die einzelnen Samples lassen sich direkt aus einem unkomprimierten .WAV Format extrahieren. Sie werden mit dem restlichen Programm des Controllers kompiliert und geflasht. Eine Momentaufnahme des PWM-Signals ist in Abb. 3.77 zu sehen.

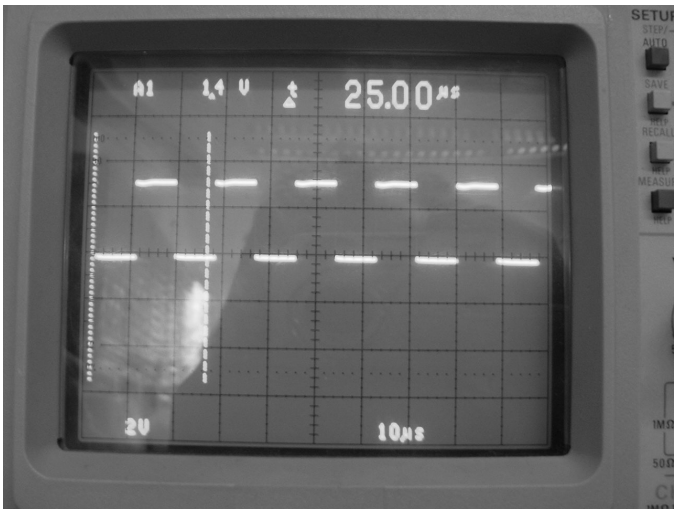


Abbildung 3.77: Resultierendes PWM-Signal am Pinout

3.4.8 Gesamtplatine

In Kapitel 3.4.6 wurden die einzelnen Komponenten des Spielfeldes vorgestellt und in Kapitel 3.4.7 die Entwicklung von Platinen zum Testen dieser Komponenten dargestellt. Die einzelnen Komponenten müssen nun zu einer Gesamtschaltung zusammengebaut werden. Grundsätzlich stehen hier zwei Möglichkeiten des Aufbaus zur Auswahl:

- **Modularer Aufbau:** Mindestens zwei Platinen mit Teilschaltungen werden erstellt. Eine Verbindung erfolgt über Stecker- und Buchsenleisten, wie zwischen den Testboards für den CPLD und das Feld. (siehe Kapitel 3.4.7)
- **Eine Platine:** Die gesamte Schaltung wird auf einer Platine untergebracht.

Der modulare Aufbau bietet insofern Vorteile als dass bei eventuellen Störungen einzelne Komponenten ausgetauscht werden können. Außerdem ist somit sichergestellt, dass genug Platz für alle Komponenten vorhanden ist. Am aussichtsreichsten wäre die Einteilung in eine Platine mit den LEDs, Fototransistoren, Tastern und dem CPLD und eine zweite Platine mit dem Mikrocontroller, der Spannungsversorgung und dem Sound.

Die Vorteile einer Platine liegen in der geringeren Arbeit für das Erstellen, da nur eine Platine gefertigt werden muss. Dadurch sind natürlich auch die Kosten für Platinenmaterial und Steckverbinder geringer, bzw. entfallen. Ein Problem stellt nur der vorhandene Platz dar. So gibt es durch die vorgegebene Positionierung der LEDs nur wenige Stellen, an denen der CPLD und der Mikrocontroller platziert werden können.

Es wurde sich dafür entschieden, nur eine Platine zu bauen. Auch wenn das Routen vielleicht mehr Zeit in Anspruch nimmt als mit zwei Platinen, so ist der Aufwand für den modularen Aufbau bei 16 zu erstellenden Feldern doch um einiges höher.

Der Schaltplan ist in Anhang A und die Layouts mit Bestückung in Anhang B zu finden. Es wurden die vorher in Kapitel 3.4.7 beschriebenen Schaltungen fast komplett übernommen. Lediglich einige zusätzliche Entstörkondensatoren für den CPLD und den Mikrocontroller wurden hinzugefügt. Außerdem fehlte die Schaltung zur Kommunikation mit anderen Feldern, wie sie in Kapitel 3.2.2 beschrieben ist. Diese wurde noch hinzugefügt.

Zum Einlöten der Leuchtdioden wurde ein Schablone erstellt, mit der auch gleichzeitig die Löcher für die LEDs und Fototransistoren in die Spielfeldfolie gestantzt werden können. Damit ist ein Übereinanderliegen gewährleistet.

3.4.9 Programmierung

Der Code für den Flash-Bootloader startet jedes Mal, wenn das System neu gestartet wird. Dies geschieht, indem der Bootloader auf den oberen Teil des RAMs abgebildet wird. Die Programmierung des Mikrocontrollers kann in zwei Arten erfolgen:

- ISP (In System Programming),
- IAP (In Application Programming)

ISP- oder IAP-Code werden im RAM gespeichert und können vom Bootloader gestartet werden.

3.4.9.1 In System Programming

Die Programmierung des Mikrocontrollers mit ISP erfolgt über einen seriellen Port (UART) und mit Hilfe des Bootloaders. Der Mikrocontroller wird zunächst in ein „Flash Modus“ gesetzt, indem der Pin “P0.14“ nach dem Reset auf LOW gesetzt wird. Dieser Zustand wird als eine externe Hardwareanfrage, die ISP Routine zu starten, interpretiert.

3.4.9.2 In Application Programming

Mit Hilfe des IAPs lässt sich aus einer Applikation der Mikrocontroller programmieren. Diese Methode hat den Vorteil, dass der Mikrocontroller auch im Betrieb programmiert werden kann, während bei der ISP-Methode der Mikrocontroller manuell in einen Flash-Modus versetzt werden muss. Durch die IAP-Methode lassen sich besonders gut mit Hilfe des Routing-Protokolls die gesamten Spielfelder schnell programmieren. Die IAP-Routine wird mit Hilfe des im Register R0 enthaltenden Zeigers, der auf den Speicherbereich zeigt, wo die Parameter sind, ausgeführt. Die Ergebnisse der IAP-Befehle werden in der Tabelle gespeichert, worauf der Zeiger im Register R1 gerichtet ist (siehe Abbildung 3.78). Die

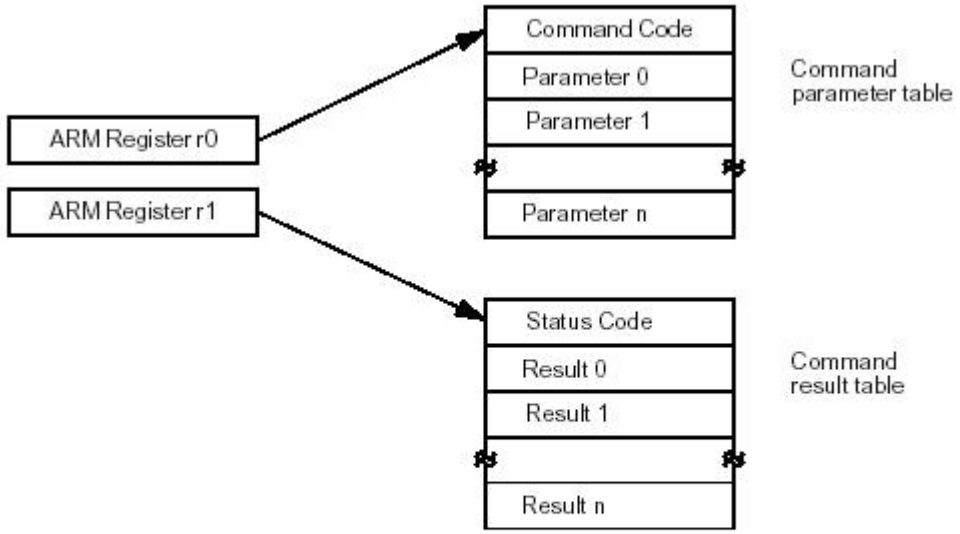


Abbildung 3.78: Pointer auf Parameter

IAP- Routine befindet sich im Register 0x7FFFFFF0 und kann wie folgt in C aufgerufen werden: Es wird zuerst definiert, wo die IAP-Routine im RAM gespeichert ist. Dies erfolgt mit dem folgenden Befehl:

```
#define IAP_LOCATION 0x7ffffff1
```

Damit die Routine ausgeführt wird, werden Datenstrukturen oder Zeiger definiert, um IAP-Befehle- und Resultat-Tabelle mit der Funktion zu verbinden. Dies wird ausgeführt, indem die folgenden Befehle geschrieben werden.

```
Unsigned long command[5];
Unsigned long result[5];
```

In der (Abbildung 3.4.9.2) werden alle Befehlscodes und Beschreibungen aufgelistet.

IAP Command	Command Code	Described in
Prepare sector(s) for write operation	50	Table 180
Copy RAM to Flash	51	Table 181
Erase sector(s)	52	Table 182
Blank check sector(s)	53	Table 183
Read Part ID	54	Table 184
Read Boot code version	55	Table 185
Compare	56	Table 186

Abbildung 3.79: Befehlstabelle

Durch die folgenden Befehle wird ein Zeiger für einen Funktionstyp mit zwei Parametern definiert, der kein Rückgabewert liefert.

```
typedef void (*IAP)(unsigned int [],unsigned int []);
IAP iap_bypointer;
```

Die Programmierung der Spielfelder erfolgt über die Flash-Routine und alle Mikrocontroller werden mit dem gleichen Code programmiert. Die Pakete werden über einen Broadcast-Mechanismus geschickt und entsprechend über das Protokoll identifiziert. Die Routine, die das Kopieren der Pakete vom RAM in den Flash ausführt, wird dann aufgerufen, wenn dieses Paket noch nicht geflasht worden ist. Das Überprüfen auf doppelte Pakete geschieht mit Hilfe der “if“ Anweisung. Die Variable “Zaehler“ wird jedes Mal inkrementiert, wenn ein Paket in den Flash kopiert wurde. Dies wird genutzt, um den Flash-Vorgang abzubrechen, wenn die gesamte Anzahl der Pakete erreicht wird. Im Array “m_pHistory“ werden die letzten fünf angekommenen Paketnummern gespeichert. Dies wird so definiert, um Duplikate zu vermeiden.

```
case FLASHPACK:
        {
            pMessageData->m_paketNr= *pMessage;
            pMessageData->m_Gesmtpack= *(pMessage+1);

    if(!( pMessageData->m_paketNr == pMessageData->m_pHistory[
            (pMessageData->m_paketNr)%5])
    {
            Zaehler++;
            pMessageData->m_pHistory[(pMessageData->m_paketNr)
                %5]= pMessageData->m_paketNr;
```

```
            flash(*(pMessage+2), pMessageData->m_paketNr,
                 Zaehler == pMessageDatam->m_Gesmtpack);
    }
    break;
        }
```

Die eigentliche Flash-Routine wird in der Funktion “flash“ ausgeführt. Diese Routine erhält drei Parameter, wobei der erste Parameter der in den Flash zu schreibenden Nachricht entspricht, der zweite Parameter gibt die gesamte Anzahl der Pakete an, und der dritte Parameter ist ein Boolescher Wert, der wahr ist, wenn alle Pakete schon angekommen sind.

```
flash(unsigned char* pMessage, Message_Data* m_paketNr, bool pKomplet)
```

Bevor die zu flashenden Daten in den Flash geschrieben werden können, muss der Bereich im Flash zuerst vorbereitet werden. Dies geschieht in den folgenden Code Zeilen. Durch die Initialisierung des Array command an Position 0 mit dem Wert 50 werden die nötigen Sektoren für die nötigen Schreiboperation vorbereitet. Die folgenden zwei Positionen des Arrays command definieren den zu präparierenden Anfangs- und Endsektor. Durch den Aufruft der Routine iap_bypointer () wird der Vorgang endgültig gestartet.

```
command[0] = 50;
command[1] = 0;
command[2] = 15;
iap_bypointer(command,result);
```

Nachdem die Parameter definiert worden sind, wird der Zeiger auf der IAP-Routine initialisiert. Für die Definition der Parameter sind folgende Angaben nötig. Der Wert 51 wird als Schreibbefehl interpretiert. Die zu beschreibende Adresse im Flash wird durch command[1] angegeben. Mit command[2] wird die Adresse des zu kopierenden Bereichs im RAM angegeben. Das Kopieren von Daten in den Flash erfolgt in Paketen von 512 Byte Größe und wird in command[3] definiert. Zuletzt wird die Takt-Frequenz des Quartz definiert

```
command[0] = 51;
command[1] = 0x40 + ((*m_paketNr-1) * 512);
command[2] = pMessage;
command[3] = 512;
command[4] = 14745;
```

Nachdem das letzte Paket angekommen ist, wird der Reset gestartet. Dieser Vorgang folgt dem gleichen Prinzip wie oben beschrieben nur mit dem Unterschied, dass in das Register 0x0 geschrieben wird, welches ein Reset als Exeption auslöst.

3.4.9.3 Spielfeldtest

Das Spielfeld besteht aus folgenden Teile, die jeweils getestet werden müssen.

- LEDs und Fototransistoren auf der oberen Platine

  Die Kommunikations-LEDs und -Fototransistoren sind am Kontroller direkt angeschlossen, die restlichen Navigations-LEDs werden über den CPLD angesteuert. Als ein allgemeines Programm zum Testen aller dieser Funktionen benutzen wird ein Programm, welches den Status der Fototransistoren wiedergibt. Dabei gehen die LEDs eines Feldes alle an, wenn **kein** Licht auf den Phototransistor fällt. Bei kompletter Lichteinstrahlung gehen alle LEDs aus. Nun kann mit z.B. der Hand teilweise das Licht unterbrechen und die LEDs reagieregn entsprechend. Bei fehlerhafter Schaltung oder insbesondere fehlerhaften lötstellen gehen einige LEDs entsprechend nicht an. Dadurch kann der Aufbau gut getestet werden.

- Kommunikation unter den Spielfeldboxen

  Die Kommunikation unter den Spielfeldboxen mit dem eigens für dieses Projekt entwickelte Kommunikationsprotokoll ergibt gute Ergebnisse. Es sind keine Fehler in der Übertragung zu erkennen. Je nach Auslastung des Controllers im realen Spielbetrieb kann sich dies aber evtl. noch ändern. Demnach muss die Übertragungsgeschwindigkeit entsprechend angepasst werden. Hier wurde nur die Kommunikation zwischen zwei Spielfeldboxen getestet und nicht das komplette Kommunikationsprotokoll inkl. Routing (Siehe dazu 4.3).

- Soundausgabe

  Die Soundausgabe lässt sich ganz einfach testen, indem ein beliebiger Sound abgespielt wird. Derzeit ist eine Soundqulität von 8kHz Samplerate und 8 Bit Sampletiefe eingestellt. Diese macht sich in der Hörqualität bemerkbar, ist aber dennoch gut erkennbar um was für einen Sound es sich handelt.

3.5 Integrationstests

Dieser Abschnitt beschreibt das Vorgehen beim Testen des Zusammenspiels zwischen Roboter und Spielfeld, da die Bestandteile dieser beiden Hauptkomponenten bereits soweit wie möglich separat getestet wurden (vgl. Abschn. 3.3.7.8 und 3.4.9.3).

Dabei wird hier vor allem der Interaktion dieser Beachtung geschenkt. Die Kommunikation, bestehend aus den Hardware-Realisierungen (vgl. Abschn. 3.3.6.3 und 3.4.6), dem Protokoll (vgl. Abschn. 3.2.3) und der Implementierung in den einzelnen Komponenten (vgl. Abschn. 3.3.7.4 und 3.4.9), ist Hauptbestandteil der Integrationstests.

3.5.1 Test der Kommunikation zwischen Roboter und Spielfeld

Das Kommunikationsprotokoll soll nach erfolgter Implementierung im Roboter und im Spielfeld unter realen Bedingungen getestet werden. Um die Datenübertragung zu testen, wird der Roboter-Prototyp auf die Test-Spielfeldbox gestellt. Dabei wird zunächst das Feld 0 (vgl. Abb. 3.70) verwendet, welches mit dem geringsten Vorwiderstand am Fototransistor beschaltet ist[17]. Zusätzlich werden an den Roboter einige frei ansteuerbare Status-LEDs angeschlossen, damit der interne Programmzustand überprüft werden kann und das Verhalten des Roboters beim Empfangen einer Nachricht besser nachvollziehbar ist. Das Spielfeld verfügt über genügend LEDs, die den Programmstatus anzeigen können.

3.5.1.1 Spielfeld sendet, Roboter empfängt

Zuerst soll die Empfangsroutine des Roboters getestet werden. Dazu sendet das Spielfeld manuell gesteuert einzelne Pakete. Anhand der Status-LEDs kann festgestellt werden, dass die Empfangsprozedur im Roboter die entgegengenommenen Nachrichten komplett verarbeitet und dass das Programm nach dem Empfang einer Nachricht wieder empfangsbereit für eine weitere Nachricht ist. Zur Überprüfung der Richtigkeit der empfangenen Daten werden an zwei Stellen Eingriffe im Programm des Roboters vorgenommen.

Zum Einen möchte soll im Fehlerfall erkennbar sein, an welcher Stelle in der Empfangsroutine der Fehler genau aufgetreten ist, um diesen bestmöglich nachvollziehen zu können. Dazu wird das Hauptprogramm des Roboters um drei Fehlercodes[18] erweitert, die angeben, welche der drei möglichen Fehlerquellen, die in Abschnitt 3.3.7.4 beschrieben werden, Ursache für die falsch empfangene Nachricht ist.

Zum Anderen soll natürlich der Inhalt der empfangenen Nachricht ersichtlich sein. Dazu werden die empfangenen Daten nacheinander in einen dafür vorgesehenen Speicherbereich des SRAM-Speichers des Controllers geschrieben, der nach erfolgter Programmausführung ausgelesen und ausgewertet werden kann (vgl.Abschn. 3.1.3.4).

Ein initialer Testlauf ergibt eine erstaunlich gute Übertragungsqualität. Nur vier der insgesamt 64 übertragenen Pakete werden fehlerhaft empfangen, alle diese haben als Ursache Timeout-Fehler beim Empfangen von Bits (Fehlercode 0xE2). Timeout-Fehler in der Initialisierungsphase (Fehlercode 0xE1) oder Prüfsummen-Fehler durch falsch empfangene Bits (Fehlercode 0xE3) kommen zunächst nicht vor. Weitere Testläufe bestätigen das anfangs erzielte gute Ergebnis. Allerdings ist zu bemerken, dass zwischen dem Senden der einzelnen Pakete eine recht große Pause von circa ein bis zwei Sekunden gemacht werden muss, da das Spielfeld die Übertragung der Daten sonst abbricht.

[17] Die einzelnen Spielfelder der Testbox erhielten alle unterschiedliche Vorwiderstände an den Fototransistoren, um den idealen Wert schließlich in einem realen Test zu ermitteln (vgl. Abb. 3.70).

[18] MESSAGE_RES_ID | 0x<code>, wobei MESSAGE_RES_ID = 0xE0

Anpassen der Empfangsgeschwindigkeiten im Roboter

Um herauszufinden, warum immer wieder vereinzelt fehlerhafte Pakete durch Timeout-Fehler beim Bit-Empfang auftreten, werden nacheinander folgende Fehlerquellen überprüft:

- Die Empfangsrate des Roboters wird überprüft, und es stellt sich heraus, dass der Periodic-Interval-Timer (PIT) des Mikrocontrollers nach dem ersten Durchlauf nicht erneut taktet. Dies ist auf ein fehlerhaftes Zurücksetzen des Timers im Programmcode zurückzuführen und kann leicht behoben werden.
- Desweiteren taktet der PIT nicht in den gewünschten Zeitintervallen, um beim Empfang eine Abtastfrequenz von 12 kHz zu garantieren. Das führt zu sehr gehäuften Übertragungsfehlern aller drei Arten (aus 64 empfangenen sind nur ein bis zwei Pakete korrekt). Der Grund dafür ist die fehlerhafte Eingangs-Taktung des Timers, die über einen Teilerfaktor mit der Frequenz der Master-Clock des Controllers zusammenhängt. Nach Messungen der Abtastrate beim Empfang mit dem Oszilloskop stellt sich heraus, dass ein Korrekturfaktor, zusätzlich zum Teilerfaktor, zur richtigen Taktung des PITs nötig ist, um die genaue Abtastfrequenz aus der Frequenz der Master-Clock zu ermitteln.
- Als letzte Fehlerquelle erweist sich die Taktung des A/D-Wandlers. Diese wurde aus konservativen und energietechnischen Gründen auf den minimal möglichen Wert gesetzt. Nach einer Anpassung auf die nötigen unteren Schwellwerte der Sample&Hold-Time und der Taktrate der Konvertierung funktioniert die Kommunikation vom Spielfeld zum Roboter fehlerfrei.

Abschließende Testläufe

Mehrere unabhängige und bei wechselnden Beleuchtungen durchgeführte Testläufe bestätigen, dass das Kommunikationsprotokoll fehlerfrei funktioniert. Alle vom Spielfeld gesendeten Pakete kommen in Wert und Reihenfolge konsistent beim Roboter an.

Der nötige große zeitliche Abstand zwischen dem Senden zweier Pakete wird von einem Fehler in der Sende-Queue des Spielfeldes verursacht, der nicht direkt mit der Implementierung des Kommunikationsprotokolles zusammenhängt. Dieser Fehler wurde behoben und die Kommunikation funktionierte daraufhin ohne Pause zwischen den einzelnen Paketen.

Durch die Umsetzung des Roboters auf andere Felder der Testbox stellte sich heraus, dass nur bei Feld 0 und Feld 1 eine initiale Synchronisation der beiden Seiten zustande kommt, also nur die Vorwiderstandswerte 1 kΩ und 2 kΩ am Fototransistor geeignet sind (vgl. Abb. 3.70 und Fußnote 17 auf S. 143). Für eine bessere Empfindlichkeit bei hellerer Belechtung wird der Widerstandswert von 1 kΩ im Spielfeld verwendet werden.

3.5.1.2 Roboter sendet, Spielfeld empfängt

Nachdem der Übertragungsweg von Spielfeld zu Roboter also verifiziert wurde, wird die Aufmerksamkeit auf die Senderoutine des Roboters gerichtet. Der Roboter sendet Datenpakete an das Spielfeld, die mit ähnlichen Methoden ausgelesen werden wie beim vorherigen Test. Hierbei ergeben sich, auch aufgrund der bereits durchgeführten Korrekturen, keinerlei Probleme in der Übertragung, so dass dieser Test zur Berechnung der Übertragungsgeschwindigkeit genutzt wird.

Messen der Übertragungsgeschwindigkeit

Zur Messung der Übertragungsgeschwindigkeit sendet der Roboter an das Spielfeld in 1024 Durchläufen jeweils zwei Datenpakete, die jeweils genau 1 Byte (8 Bits) groß sind. Die für das Übertragen benötigte Zeit wird gemessen, um die Übertragungsgeschwindigkeit und die damit verbundene Nutzdatenrate zu bestimmen.

Der gesamte Sende- und Empfangsdurchlauf dauert in etwa 64 Sekunden, so dass sich eine Übertragungsgeschwindigkeit von ungefähr 2 kBytes pro Minute oder circa 32 Bits pro Sekunde ergibt. Diese doch recht geringe Datenrate ergibt sich aus der zeitaufwändigen Synchronisationsprozedur im Kommunikationsprotokoll (vgl. Abschn. 3.2.3), die vor dem Übertragen eines Pakets stattfindet. Diese ist jedoch tolerabel und sogar notwendig, da die Steuerbefehle vom Spielfeld an den Roboter jeweils nur 1 Byte groß sind (7 Bits Befehlsbreite und ein Paritätsbit, vgl. Abschn. 3.3.7.4), und der Roboter im Allgemeinen nach dem Empfang eines Befehles das Feld verlässt. Zwischen dem Roboter und dem erreichten Feld ist dann eine erneute Synchronisation erforderlich.

3.5.2 Test der Befehlsausführung im Roboter

Zum Abschluss der Testreihe zur Kommunikation wird nun die Nachrichtenauswertung im Roboter getestet, also ein kompletter Kommunikationszyklus im späteren Spielverlauf simuliert. Hierzu sendet das Spielfeld den Steuerbefehl 0x22 an den Roboter, der bewirken soll, dass der Roboter ein Feld vorwärts fährt (vgl. Abschn. 3.3.7.1). Im Gegensatz zu den vorherigen Tests, in denen die Motoren aus Energiespargründen nicht angeschlossen waren, werden diese hier wieder mit Strom versorgt. Deshalb können wir die Reaktion des Roboters, die er auf den Empfang des Steuerbefehls zeigt, beobachten. Wie erwartet, bewegt der Roboter sich vorwärts, die Räder rutschen jedoch auf dem Makrolon, so dass er das nächste Spielfeld nicht erreicht. Wenn der Roboter per Hand auf das Ausgangsfeld zurückgesetzt wird, ist er in der Lage, den nächsten Befehl zur Vorwärtsbewegung direkt entgegenzunehmen und auszuführen. Einer kompletten Simulation eines Spielzuges und einem Test der Fehlerkorrekturalgorithmen (vgl. Abschnitt 3.3.7.7) steht somit nur noch das Rutschen der Räder im Weg, für das eine Lösung in Abschnitt 3.3.3.4 beschrieben wird.

3.6 Host-Interface

Der Anschluss des Host-PC an die Spielfeldboxen soll über eine serielle Verbindung realisiert werden. Da der Aufbau des Spielfeldes aus den Spielfeldboxen variabel gehalten werden sollte, muss somit jede Spielfeldbox an jeder der vier Seiten eine Möglichkeit bieten, eine serielle Verbindung zum Host herzustellen. Da aber die Verbindung der Spielfeldboxen untereinander über Kontaktflächen (siehe Abschnitt 3.4.3.1) geschieht und auch die Konstruktion mit den Winkeln (siehe 3.4.3.2) wenig Raum für Anschlüsse lässt, wurde die Verbindung über einen Adapter gewählt.
Dieser Adapter besitzt auf der einen Seite die Möglichkeit, eine Verbindung über ein serielles Kabel mit der entsprechenden Schnittstelle des PC herzustellen. Auf der anderen Seite kann er mit einer Spielfeldbox verbunden werden.

3.6.1 Mechanischer Aufbau

Es gibt mehrere Aspekte, die den Aufbau des Adapters beeinflussen. Zum einen ist das der Aufbau der Spielfeldboxen und zum anderen auch die Verbindung zum Rechner.
Da die Spannungsversorgung der Spielfeldboxen über die Winkel (siehe Abschnitt 3.4.3.2) geschieht, enthält der Aufbau des Adapters zwei Winkel, die wie im folgenden noch erläutert ein wenig angepasst werden. Außerdem hat der Adapter den gleichen Aufbau der Kommunikationsverbindungen.

Daraus resultiert der Aufbau des Adapters aus zwei Winkeln ähnlich einer halben Spielfeldbox und der Verbindung der beiden Winkel durch eine Platine. Diese Platine besitzt auf der Außenseite die zwei Kontaktflächen und die zwei Federn (siehe Abschnitt 3.4.3.1) zur Verbindung mit dem Feld. Für die Verbindung zum Rechner enthält der Adapter eine Sub-D-Buchse, um ein serielles Kabel anzuschließen.

Ein weiterer Aspekte ist die Unterbringung der Elektronik. Eine Platine ist im Aufbau schon enthalten, die jedoch auf der einen Seite durch die Kommunikationsverbindungen und an den Rändern durch die Winkel belegt ist. Es bestand die Möglichkeit eine weitere Platine vorzusehen, um eine Erleichterung des Layoutroutings durch eine zweiseitige Platine zu erreichen. Es gibt aber das Ziel, den Adapter so kompakt wie möglich und auch möglichst kostengünstig zu halten. Daher ist die Elektronik auf der anderen Seite der schon vorhandenen Platine untergebracht. Diese Reduktion des benötigten Platzes bietet die Möglichkeit, die beiden Winkel jeweils auf einer Seite zu kürzen.

Die Verkleidung der noch offenen Seiten außer der Unterseite soll durch dünne Makrolonplatten, die auch im Feldaufbau Verwendung finden, erreicht werden. So ergeben sich folgende Maße: Der Adapter ist wie die Spielfeldboxen 5 cm hoch, 15 cm breit, aber nur rund 1 cm tief.

3.6.2 Elektronik

Die Elektronik enthält in erster Linie natürlich den Mikrocontroller LPC2124 [Phi03] von Philips, welcher auch in der Elektronik der Spielfeldboxen Anwendung findet. Für diesen wird ein Quarz mit 14,7456 Mhz eingesetzt.

Die Spannungsversorgung gestaltet sich ähnlich der in den Spielfeldboxen. Der Unterschied zur Elektronik in den Spielfeldboxen ist das nur einseitige Routing und die nur einseitige Bestückung mit Bauelementen.
Der bedrahtete Brückengleichrichter FBU4A [Dio05] wurde durch das Zurechtbiegen der Beine und das flach auf die Platine Legen vom Prinzip her in ein SMD-Bauteil gewandelt. So konnte der selbe Gleichrichter wie im Spielfeld verwendet werden.
Die Spannungsregler sind jene, welche beim Test der Spannungsversorgung (beschrieben in 3.4.7.4) für das Feld benutzt wurden. Zum einen ist das die SMD-Variante des LM1086 [Nat05] und der 1,8 V Regler XC6201P162MP [Tora]. Die in den Datenblättern vorgeschlagenen Kondensatoren für die Spannungsregler sind ebenfalls SMDs.

Für die Kommunikation mit dem Feld wurden zwei Pins des Mikrocontrollers vorgesehen. Dabei besitzt einer Interruptfähigkeit für die Clockleitung der Kommunikation, und der andere ist ein einfacher IO-Pin für die Daten. Vor jeden der beiden Kommunikationspins ist ein Widerstand von 51 Ω geschaltet und danach ein Pull-Up-Widerstand von 4,7 kΩ. Außerdem ist zum Schutz der Pins jeweils eine Diode in Sperrrichtung zu Masse und eine in Durchlassrichtung zur Versorgungsspannung geschaltet.
In dem Schaltplan und im Layout im Anhang (6.7 und 6.28) sind mehrere 0 Ω Widerstände erkennbar. Diese dienen nur als Brücken im Layout der Schaltung.

Zur Kommunikation mit dem PC wird die UART0 Schnittstelle des Mikrocontrollers benutzt. Dabei wurden nur die Pins RxD0 (Seriell Input) und TxD0 (Seriell Output) benutzt. Die Pegel des Mikrocontrollers werden durch den RS232-Pegelwandler ICL3232CB [Int02] umgesetzt.

Die serielle Schnittstelle wird zusätzlich zur Programmierung des Mikrocontrollers benutzt. Dazu ist ein SMD-Jumper vorgesehen, mit dem zwischen ISP (In-System Programming) Modus und normalem Betrieb umgeschaltet werden kann. Weiterhin ist eine Stiftleiste mit zwei Pins (Masse und RST) für die Beschaltung mit einem externen Resettaster vorgesehen. Ansonsten wird der Reset beim Anlegen der Versorgungsspannung durch einen Kondensator zwischen dem RST Pin und Masse generiert.

3.6.3 Programmierung

Die Programmierung geschieht wie erwähnt über die serielle Schnittstelle. Dabei soll das Progamm zum einen aus der Implementierung der seriellen Schnittstelle des Mikrocontrollers bestehen, zum anderen beinhaltet das Programm das Kommunikationsprotokoll für die Spielfeldboxen. Dadurch sollen Nachrichten, die vom Host-PC empfangen werden, an die Spielfeldboxen weitergegeben werden. Aber auch die andere Richtung vom Spielfeld zum PC soll möglich sein (siehe auch 3.2.2).

4 Software

4.1 Einleitung

In diesem Kapitel wird die Software beschrieben, die bei R_3D_3 zum Einsatz kommt. Zunächst werden kurz die Anforderungen aufgelistet, die zum Design der eingesetzten Architektur geführt haben und ein Überblick über diese Architektur gegeben.

Dann folgt eine detailierte Beschreibung des entwickelten Protokoll-Stacks, der für die Kommunikation der Komponenten untereinander nötig ist. Dabei wird zuerst auf den Aufbau des Stacks und seine Bestandteile eingegangen, bevor diese im Einzelnen genau erleutert werden.

Abschließen wird der Spielablauf der Software in den Host-Systemen erläutert und ein Überblick über die bereits entwickelten GUI-Elemente gegeben.

4.2 Architektur

4.2.1 Anforderungen

Wichtig für das Design der Softwarearchitektur war, dass die Struktur nach Möglichkeit in allen Komponenten wieder verwendet und flexibel angepasst werden kann. Komponenten, die mit Software ausgestattet werden müssen, sind:

- Die Roboter,
- die Spielfelder,
- der Parkplatz,
- die Host-PCs und
- die Host-Spielfeld Schnittstellen.

Jede dieser Komponenten hat, bedingt durch die jeweilige Hardware, ganz eigene Anforderungen an die Software. Während in den Robotern wenig Speicher und Rechenleistung zur Verfügung steht und die Aufgaben der Software sehr stark begrenzt sind, läuft die Host-Software auf gewöhnlichen PCs und hat eine deutlich größere Anzahl an Aufgaben zu bewältigen.

Daher ist die Software-Architektur sehr modular aufgebaut, so dass die Software für die einzelnen Komponenten und ihre entsprechende Hardware-Architektur mit minimalem Aufwand zusammengestellt werden kann.

4.2.2 Überblick

Die gesamte Software-Architektur (Abb. 4.1) kann grob in zwei Bereiche eingeteilt werden:

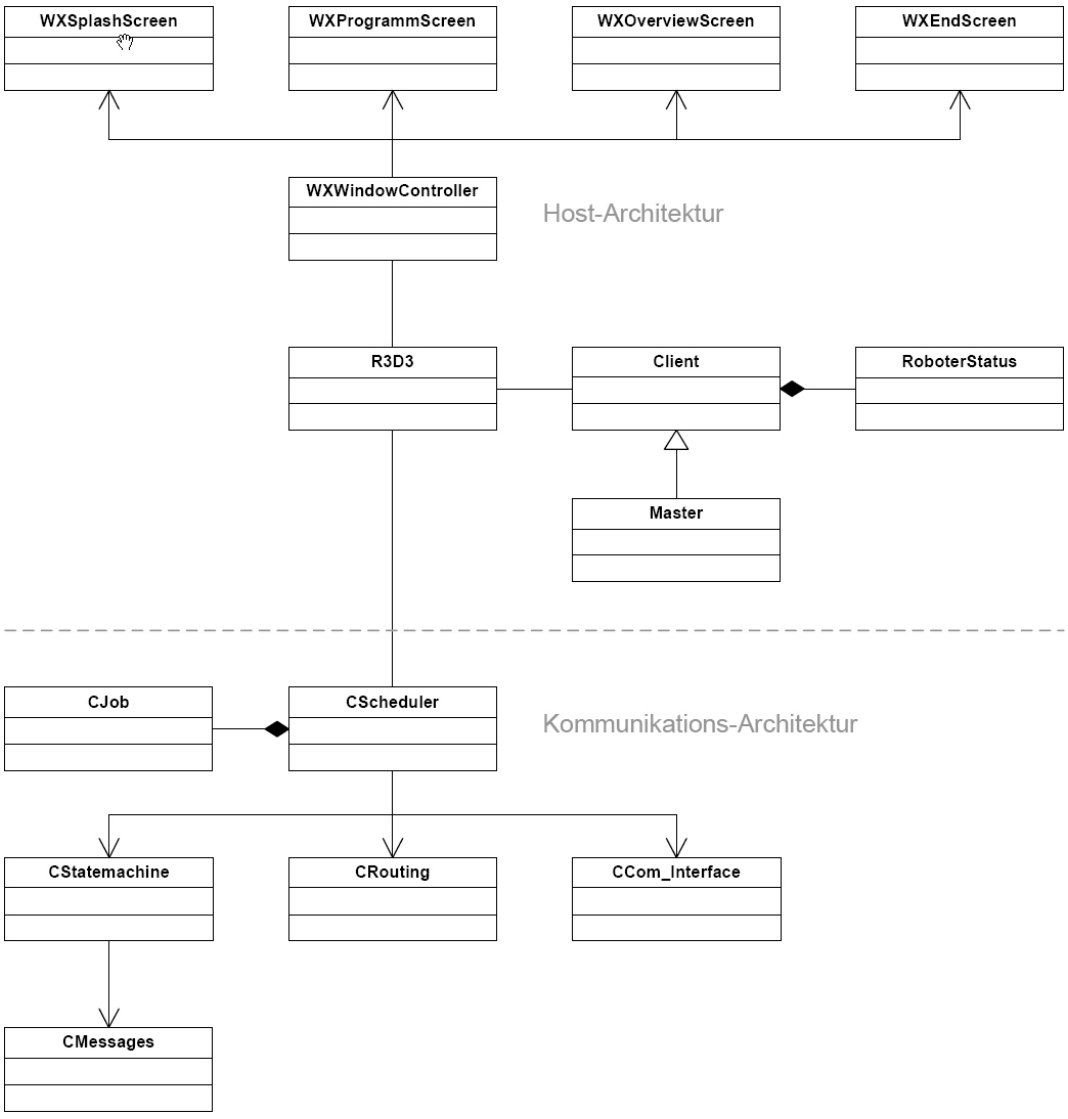


Abbildung 4.1: Architekturüberblick

- Die Host-Architektur, die nur in den Host-PCs zum Einsatz kommt und
- die Kommunikations-Architektur, die zumindest in Teilen in allen Komponenten zu finden ist.

Die Kommunikations-Architektur bildet die Grundlage für den Nachrichtenaustausch zwischen den Komponenten. Dieser Teil der Architektur wird in allen Komponenten in entsprechend modifizierten oder eingeschränkten Varianten wieder verwendet. Der in dieser Architektur verwendete Protokoll-Stack wird in Abschnitt 4.3 ausführlich beschrieben.

Die Host-Architektur umfasst sowohl die GUI-Klassen, die für die Interaktion des Spieles mit dem Benutzer verantwortlich sind und in Abschnitt 4.4.4 beschrieben sind, als auch die Strukturen für die Applikationslogik, die in den Hostsystemen abläuft. Die R3D3-Klasse bildet dabei das Herzstück und den zentralen Einstiegspunkt[1] der Software für den Host-PC. Hier werden auch die Informationen der Spieler verwaltet, wie zum Beispiel der Status der jeweiligen Roboter. Da ein Host-System zusätzliche Aufgaben zur Synchronisation und Konsistenzerhaltung übernehmen muß, wird einer dieser Hosts als Master ausgezeichnet. Seine Aufgabe besteht darin, als Spielleiter zu fungieren und die einzelnen Phasen des Spielablaufs (Kartengeben, Abarbeiten von Befehlen und Fabrikelementen) anzustoßen. Alle anderen Hosts arbeiten als Clients. Diese synchronisieren alle Aktionen, die die Spielkonsistenz beeinträchtigen würden, zunächst mit dem Master. Alle anderen Aufgaben verarbeiten die Hosts jedoch völlig dezentral.

Wie die genaue Schnittstelle zwischen der Host- und der Kommunikations-Architektur aussehen wird, ist noch nicht spezifiziert. Geplant ist bisher für den Host eine weitere Ausprägung der Jobs zu schaffen und so über den Scheduler (vgl. 4.3.4) die nötigen Informationen zwischen den Schichten der Architektur auszutauschen. Die genauen Details werden zusammen mit den Details der Applikationslogik realisiert.

4.3 Protokoll-Stack

4.3.1 Einleitung

Wie im vorhergehenden Kapitel 4.2.1 bereits gezeigt wurde, besteht das R_3D_3-Projekt aus verschiedenen Komponenten, die mittels Kommunikation miteinander interagieren. Für diese Kommunikation sind Protokolle nötig, die die Semantik und das Format der auszutauschenden Informationen festlegen.

Wie das ISO/OSI Modell zeigt, hat es sich bewährt Protokolle in Schichten zu modellieren und zu einem so genannten Protokoll-Stack zusammenzufassen. Jede Schicht, auch Layer genannt, benutzt dabei zur Erfüllung ihrer speziellen Aufgabe die jeweils nächst tiefere Schicht im Protokollstack.

[1] In allen anderen Komponenten liegt der Einstiegspunkt im Scheduler

Der R3D3-Protokollstack hat den folgenden dreistufigen Aufbau:

RoboRally-Kommunikation: Auch als HighLevel-Layer bezeichnet: Auf dieser Ebene werden Nachrichten verarbeitet, die für den Spielbetrieb nötig (Befehle an den Roboter, Fabrikelemente usw.) sind.

Routing-Protokoll: Der Routing-Layer ist dafür zuständig Nachrichten vom Sender an den richtigen Empfänger weiter zu leiten.

Low-Level Netzwerk-Protokoll: Auf dieser Ebene werden Bits physikalisch übertragen und sichergestellt, dass der Empfangspartner diese korrekt empfängt.

Nicht alle Komponenten haben allerdings den gleichen Protokoll-Stack: Beim Roboter wurde zum Beispiel auf den Routing-Layer verzichtet, da die Kommunikation zwischen Host-PC und Roboter über das Spielfeld abläuft. Jeder Roboter kommuniziert also nur direkt mit dem Spielfeld, auf dem er steht, so dass in diesem Fall ein Routing von Nachrichten überflüssig ist.

Der Host nimmt ebenso eine Sonderstellung ein: Er wird über eine RS232-Schnittstelle an das Spielfeld angeschlossen, die aber inkompatibel zum Low-Level-Protokoll der Spielfelder ist (vgl. 3.2.1). Aus diesem Grund gibt es eine Host-Feld-Schnittstelle, mit deren Hilfe der Host mit dem Feld kommunizieren kann.

In den weiteren Kapiteln werden die einzelnen Protokollschichten näher erleutert. Der Low-Level-Layer wurde bereits in 3.2.3 beschrieben, so dass hier jetzt noch der Routing-Layer und die RoboRally-Kommunikation näher erleutert werden müssen.

4.3.2 Routing Protokoll

4.3.2.1 Allgemeine Aufgaben

Im Allgemeinen bezeichnet Routing das Finden eines Weges durch ein Netzwerk, bei gegebenem Start- und Zielknoten. Die meisten Routing-Protokolle kommen im Internet zum Einsatz, weil es sich um das größte und bekannteste Netzwerk handelt. Diese Protokolle müssen allerdings auf ihre Verwendbarkeit geprüft werden, da an das Routing-Protokoll von R3D3 einige zusätzliche Anforderungen gestellt werden:

- **Roboter-Orientierung:** Für die Abarbeitung von Befehlen (zum Beispiel ein Feld vorwärts) muss für Feld und Roboter eine Kodierung für Position und Blickrichtung gefunden werden. Nur so kann das Spielfeld herausfinden, wo die neue Position des Roboters nach der Befehlsverarbeitung ist, und welche Eigenschaften das neue Feld hat (Grube, Brenner, etc.)
- **Host-Anschluss:** Der Host-PC muss die gleichen Kommunikationsprotokolle implementieren wie die Felder untereinander, damit eine Verständigung möglich ist. Ebenso muss das gleiche Protokoll auf die in 1.4 als Erweiterungsmöglichkeit an-

gegebenen Konsolen übertragbar sein. Das bedeutet vor allem, dass der Code portierbar sein muss.

- **Selbst-Organisation:** In einer so genannten Startup-Phase soll sich das Spielfeld selbstständig eine Struktur aufbauen, mit deren Hilfe nicht nur das Routing von Nachrichten möglich ist sondern auch die oben erwähnte Orientierung.

4.3.2.2 Klassifizierung von Routing-Protokollen

Allgemein wird zwischen *statischen* und *dynamischen* Routing-Protokollen unterschieden. Beim *statischen* Routing wird angenommen, dass die Netztopologie keinen Änderungen unterliegt, so dass jedem Router die gesamte Topologie per Konfiguration vorliegt. Das führt dazu, dass bei jeder Änderung die Routing-Tabellen manuell angepasst werden müssen. Bei Topologieänderungen kann das System sich daher nicht selbst reorganisieren.

Beim *dynamischen* Routing wird durch ständiges Update der Status der Routen überprüft und bei Ausfall eine andere genommen, so dass kein manueller Eingriff erforderlich ist. Da die Struktur eines R3D3-Spielfeldes nicht a priori vorgegeben ist, sondern vielmehr frei zusammensteckbar sein soll, kommen hier nur dynamische Protokolle in Frage. Die größte und meist genutzte Unterkategorie der dynamischen Routing-Protokolle bilden die **Shortest-Path-Protokolle**.

Dabei ist es das Ziel, eine Nachricht kostengünstig durch das Netzwerk zu transportieren, wobei jede Kante des Netzes gewisse Kosten verursacht. Haben alle Kanten die Kosten Eins, handelt es sich um *Minimum-Hop-Routing*. Die bekanntesten und auch im Internet am häufigsten verwendeten Algorithmen zur Berechnung des kürzesten Weges sind Link-State-Algorithmen und Distance-Vector-Algorithmen, die in den folgenden Abschnitten näher beschrieben werden.

Link-State

Bei einem Link-State-Algorithmus sendet jeder Knoten Informationen über seinen sog. Link-State an alle anderen Knoten. Unter dem Link-State versteht man Informationen über die Verbindungen mit anderen Knoten und die Kosten, eventuell Bandbreite und Auslastung dieser Verbindungen. Aus diesen Informationen kann jeder Knoten einen Graph der Netzwerk-Topologie generieren und anhand dieses Graphen Routing-Entscheidungen treffen, wodurch Routing-Schleifen verhindert werden.

Die Mehrheit der Routing-Protokolle ist dieser Klasse zu zuordnen. Das bekannteste und im Internet verbreitetste Protokoll ist OSPF (Spezifikation in [CS05]). Allerdings haben Link-State-Protokolle einen hohen Speicherplatz und hohe Prozessorlast, so dass sie sich nicht für die Implementierung auf einem Mikrokontroller eignen. Deswegen wird hier nicht weiter darauf eingegangen.

Distance-Vector

Distance-Vector-Algorithmen treffen Routing-Entscheidungen nur anhand von Kosten, die einem Router von seinen Nachbarn zugeschickt werden. Es wird der Nachbar mit den geringsten Kosten zum Ziel als nächster Hop in der Routing-Tabelle vermerkt. Bei Änderungen der Netztopologie und nach einem festgelegten Zeitintervall findet ein Update mittels eines Broadcasts statt (dabei wird die gesamte Routing-Tabelle an alle Nachbarn gesendet).

Obwohl Distance-Vector Algorithmen weniger Rechenzeit brauchen, ist auch hier die Verwendbarkeit nicht optimal, wie im folgenden Abschnitt erläutert wird. Weitere Routing-Protokolle in [Sch87].

Allgemeine Probleme

Das Hauptproblem der hier vorgestellten Routing-Protokolle ist, dass sie keine geeignete Struktur für die Anforderung der Roboter-Orientierung aufweisen. Das kann nur durch Speichern zusätzlicher Informationen kompensiert werden, wie an dem folgenden Beispiel illustriert werden soll:

Jede Spielfeldbox bekäme eine ID, wobei die Vergabe über Hardware oder Software erfolgen könnte. Die IDs der jeweiligen Spielfeldbox in Blickrichtung, die linke, rechte und die ID der Spielfeldbox entgegen der Blickrichtung eines jeden Roboters müssten zusätzlich abgespeichert werden. Nur so kann der neue Standort eines Roboters nach einer Befehlsverarbeitung ermittelt werden.

Daher eigenen sich diese Protokoll nur bedingt. Ein geeigneter Ansatz, der ohne zusätzliche Speicherung auskommt, wird im folgenden Abschnitt vorgestellt.

4.3.2.3 Verwendung von Peer-to-Peer-Technik

Die bisher vorgestellten Routing-Protokolle basieren auf einer vorgegebenen Hierarchie von Routern und Netzen. Aber eine solche Hierarchie existiert bei einem so homogenen Feld wie bei R_3D_3 nicht. Daher lohnt sich ein Blick auf P2P-Netze, weil besonders bei diesen eine völlig dezentrale Struktur und Organisation stattfinden muss. Dabei bietet sich vor allem das Content Adressable Network (CAN) an:

Dieses Netz teilt seine Peers in einen logischen Koordinatenraum ein, in dem jeder Peer seine eigene Zone besitzt. Nachrichten, die durch das CAN geroutet werden, haben als Zieladresse ein Koordinatenpaar, so dass sie mittels eines einfachen Greedy-Algorithmus weitergeleitet werden können. Liegen die Zielkoordinaten innerhalb des eigenen logischen Raumes, wird die Nachricht bearbeitet. Weitere Informationen in [LCP+04].

Auf R_3D_3 übertragen bedeutet das, dass die Spielfelder in ein virtuelles Koordinatensystem eingeteilt werden. Jedes Spielfeld bekommt ein eigenes, eindeutiges Koordinaten-

paar bestehend aus x- und y-Koordinate zugeordnet. Die Bestimmung und Festlegung dieser Koordinaten muss in einer sog. Init-Phase nach dem Einschalten stattfinden.

Das Routing kann mit Hilfe des euklidischen Abstandes durchgeführt werden: Eine Spielfeldbox speichert die Koordinaten der Nachabrn, so dass bei einer eintreffenden Nachricht anhand der Zieladresse in der Nachricht (bestehend aus x,y Koordinate) den Abstand zum Ziel berechnen kann. Gesendet wird die Nachricht an den Nachbarn mit dem kleinsten Abstand.

4.3.2.4 Das R_3D_3-Routing-Protokoll

In diesem Abschnitt wird das Routing-Protokoll von R_3D_3 vorgestellt. Es verwendet die bereits beschriebene koordinatensystembasierende Struktur, in der das Routing mittels euklidischem Abstand erfolgt. Die Implementierung des Protokolls wurde in C vorgenommen, so dass sie problemlos auf der Zielhardware zum Einsatz kommen kann.

Die Abbildung 4.2 zeigt den Header des Routing-Layers, in dem alle Funktionsdeklarationen stehen. Die drei obenstehenden Funktionen CRouting_getAdress, CRouting_setAdress, CRouting_ErrorFromComLayer stehen als Interface für die Interaktion mit dem übergeordneten bzw. untergeordneten Layer zur Verfügung:

- CRouting_getAdress wird aufgerufen, wenn der Low-Level-Layer eine Nachricht empfangen hat. Die Funktion beginnt damit, die Nachricht zu parsen und ihre Daten in dem Datentyp struct Routing_Message zu speichern, den die weiteren Funktionen als Eingabeparameter akzeptieren. Abhängig von Nachrichtentyp und Koordinaten, wird folgendes getan:

  1.) Init-Nachricht: Wenn noch nicht initialisiert, CRouting_init aufrufen, sonst CRouting_UpdateHopCounter

  2.) BroadcastNachricht: CRouting_HandleBroadcastMessage wird aufgerufen

  3.) Zielkoorindaten liegen in eigenem Koordinatenraum: Nachricht an den übergeordneten Layer weiterleiten

  4.) sonst Nachricht an Nachbarn mit geringstem Abstand zum Ziel weiterleiten (CRouting_sendToNearestNeighbor)

- CRouting_setAdress wird aufgerufen, wenn der übergeordnete Layer eine Nachricht schicken möchte. Dabei wird nach folgenden Kriterien gearbeitet:

  1.) Nachricht liegt in eigenem Koordinatenraum, also wieder an den oberen Layer zurücksenden

  2.) Nachricht ist ein Broadcast: Aufrufen von CRouting_broadcastMessage

  3.) sonst CRouting_sendToNearestNeighbor

- CRouting_ErrorFromComLayer wird aufgerufen, wenn auf dem untergeordneten Layer ein Fehler aufgetreten ist. Der entsprechende Nachbar wird als unerreichbar vermerkt. Eine erweiterte Fehlerbehandlung wird nächstes Semester eingeführt.

```
struct Routing_Message {
  char m_broadcast;
  char m_xcoordinate;
  char m_ycoordinate;
  char m_Type;
  char m_initXcoordinate;
  char m_initYcoordinate;
  char m_PositionBit;
  char m_IncrementBit;
  char m_HopCounter;
  unsigned char m_nMessageID;
  unsigned char m_MessageDatabuffer[100];
  unsigned char m_MessageDatabuffer_Size;
};

char CRouting_getAdress(Job* pJob);
char CRouting_setAdress(Job* pJob);
char CRouting_ErrorFromComLayer(Job* pJob);

char CRouting_init(Job* pJob, Routing_Message* pMessage);
char CRouting_UpdateHopCounter(Job* pJob,
  Routing_Message *pRouting);

char CRouting_HandleBroadcastMessage(Job* pJob,
  Routing_Message* pMessage);

char CRouting_sendToNearestNeighbor(char recievedFrom,
  Routing_Message* pMessage);

char CRouting_broadcastMessage(char recievedfrom,
  Routing_Message* pMessage);
```

Abbildung 4.2: Datei: Routing.h

Init-Phase

In der Init-Phase wird die Koordinatensystem-Struktur aufgebaut. Damit dies geschehen kann, müssen sowohl die Felder der Spielfeldbox als auch die Nachbarn unabhängig von der Koorinatenzuweisung durchnummeriert werden. Die genaue Belegung der Felder mit Indices und die Nummerierung der Nachbarn ist in 3.4.6 (Abb. 3.70) bereits angegeben worden.

Die Belegung der Felder mit den Koordinaten geschieht wie folgt: Der Parkplatz weist sich selbst die Koordinaten (0, 0) zu und beginnt darauf, die anderen Spielfeldboxen – entsprechend ihrer Position – die Koordinaten zuzuweisen. Dazu legt er zuerst die Orientierung fest. Die positive x-Achse zeigt nach rechts (Richtung seines Nachbarn Eins) und die positive y-Achse nach oben (in Richtung Nachbar Null). Danach ordnet jede Spielfeldbox ihren neun Feldern ihren Indices entsprechend die Koordinaten zu.

Dafür ist die Funtion CRouting_init verantwortlich, die zuerst alle Informationen aus der Nachricht extrahiert (Für das genaue Format der Init-Nachricht siehe Tabelle 4.1). Aus der gesendeten Koordinate und der zwei Orientierungsbits kann die Funktion die Felder mit Koordinaten belegen.

Da das System auch mit nicht-rechteckigen Spielaufbauten zurechtkommen muss, wird zur Initialisierung ein einfaches Broadcast-System verwendet. Das bedeutet, dass der Parkplatz beginnt, seine Spielfeldboxen, an die er angeschlossen ist, zu initialisieren, indem er eine Init-Nachricht schickt. Diese Spielfeldbox sendet den Befehl an alle seine Nachbarn weiter usw. Die Orientierungsbits geben dabei an, welche Koordinate sich ändert, und ob sie erhöht oder vermindert wird. Ihre Belegung und die entsprechende Bedeutung ist der folgenden Tabelle zu entnehmen.

1. Bit	2. Bit	Beschreibung
0	0	x-Koordinate wird verringert
0	1	x-Koordinate wird erhöht
1	0	y-Koordinate wird verringert
1	1	y-Koordinate wird erhöht

Zusätzlich zu den Koordinaten speichert jede Spielfeldbox einen so genannten HopCounter, der die Entfernung zum Parkplatz angibt. Dies wird dazu benötigt, um bei einem nicht-rechteckigen Spielaufbau den Roboter zum Parkplatz zurückzuleiten. Der aktuelle HopCounter wird in die Init-Nachricht aufgenommen, den jede Spielfeldbox speichert, um eins erhöht und weiterschickt. In der Startup-Phase kommt es daher häufig vor, dass zusätzliche Init-Nachrichten geschickt werden müssen, um die HopCounter aller Spielfeldboxen richtig zu belegen. Eine Spielfeldbox, die eine weitere Init-Nachricht empfängt, ruft ihre CRouting_UpdateHopCounter-Funktion auf, die die folgenden Fälle bearbeitet:

1. **Fall:** Eigener Hop = Hop in Nachricht ⇒ HopCounter für Nachbarn, von dem die Nachricht erhalten wurde auf (Hop in Nachricht-1) setzen

2. **Fall:** Eigener Hop < Hop in Nachricht ⇒ Der Sender der Nachricht hat einen zu hohen Hop, also wird ihm der (Eigene Hop+1) zugesendet

3. **Fall:** Eigener Hop > Hop in Nachricht ⇒ Eigener Hop ist zu groß, daher wird Eigener Hop = Hop in Nachricht gesetzt und allen anderen Nachbarn (EigenerHop+1) zugesendet.

Broadcast-Nachrichten

Das R3D3-Routing-Protkoll kann Nachrichten in zwei Arten weiterleiten. Nachrichten, die an einen Empfänger gehen, werden von der Funktion CRouting_SendTonearestNeoghbor nach dem mehrfach beschriebenen Verfahren weitergeleitet. Nachrichten, die an alle Empfänger gehen sollen, werden durch einen sog. Broadcast an alle gesendet.

Eine empfangene Broadcast-Nachricht wird von der Funktion CRouting_HandleBroadcastMessage bearbeitet, die überprüft zuerst anhand der eindeutigen Message-ID, ob diese Nachricht bereits einmal empfangen worden ist, und verwirft sie in dem Fall. Als nächstes wird geprüft, ob die Adresse (bestenhend aus einem Koordinatenpaar) der BROADCASTADRESS entspricht. BROADCASTADRESS ist definiert als `((char)0xC0)` also -63 und ist so gewählt, dass sie während des Spiels nicht vorkommen kann. Diese Adresse wird gebraucht, um die beiden verschiedenen Broadcast-Modi auseinander zu halten.

- **Broadcast**, der an *alle* Spielfelder gehen soll. Die CRouting_HandleBroadcast Funktion leitet die Nachricht an den übergeordneten Layer weiter und sendet sie zusätzlich an alle Nachbarn weiter. Das Adressfeld der Nachricht wird mit der oben erwähnten BROADCASTADRESS belegt.

- **Broadcast**, der aufgrund eines *Routing-Fehlers* passiert: Wenn eine Spielfeldbox eine kleinere Distanz zum Ziel hat als alle ihre erreichbaren Nachbarn, was bei nicht-rechteckigen Konstruktionen oder bei Konstruktionen mit einem "Loch" der Fall sein kann, dann kann sie die Nachricht nur ans Ziel bringen, indem sie einen Broadcast schickt. Bei diesem Broadcast wird die Zieladresse ins Adressfeld der Nachricht eingetragen, so dass die Zielbox diese Nachricht an seinen oberen Layer weiterleitet, alle anderen Boxen diese Nachricht aber **nicht** an ihren oberen Layer weitergeben, sondern die Nachricht **nur** weiterleiten. Das bedeutet, dass alle Spielfeldboxen die Nachricht weiterleiten, denn sie ist ja eine Broadcast-Nachricht, aber diese nicht bearbeiten.

CRouting_HandleBroadcast benutzt daher in jedem Fall die Funktion CRouting_broadcastMessage, die eine übergebene Nachricht an alle verfügbaren Nachbarn sendet, aber abhängig vom Broadcasttyp wird die Nachricht nicht an den oberen Layer weitergegeben.

4.3.2.5 R_3D_3-Routing-Protokoll Tests

Die Tests für das Routing-Protokoll fanden nicht auf der Zielhardware sondern in einer auf Sockets basierenden Umgebung statt. Dadurch ließ sich das Protokoll sehr komfortabel und schnell debuggen. Die Test-Umgebung sieht so aus, dass die Low-Level Kommunikation durch einen Thread simuliert wurde, der auf einem Socket lauscht. Wird etwas empfangen, wird der Routing-Layer über den Scheduler (siehe 4.3.4) aktiviert und kann die empfangene Nachricht bearbeiten.

Jede Spielfeldbox bekommt vier Sockets entsprechend der Anzahl seiner Nachbarn. Da die Sockets im verbindungsorientierten Modus arbeiten, müssen die Spielfeldboxen in einer bestimmten Reihenfolge aktiviert werden, so dass sich je ein verbindender und ein lauschender Socket miteinander verbinden. Danach kann der Parkplatz mit der Initialisierung der Koordinaten beginnen.

Es wurden insgesamt fünf Tests mit bis zu acht Spielfeldboxen ausgeführt, die zu verschiedensten Anordnungen kombiniert wurden. So sollten insbesondere nicht-rechteckige Spielfelder und solche mit einem “Loch“ getestet werden, um die in 4.3.2.4 erwähnten Routing-Fehler zu simulieren. Die Tests wurden erfolgreich abgeschlossen, so dass als nächste Schritte die Ausführung auf dem Mikrokontroller durchzuführen ist.

4.3.3 High-Level Protokoll

4.3.3.1 R_3D_3-Nachrichten

Das High-Level Protokoll bildet die oberste Schicht des Protokoll-Stacks. Auf dieser Ebene werden die eingegangenen Nachrichten entsprechend eines internen Zustandes verarbeitet, indem bestimmte Aktionen ausgelöst werden. Dieses Verhalten wird als Zustandsübergangssystem modelliert, die genaue Beschreinug dazu ist in 4.3.3.2 zu finden.

Die Nachrichten sind in Tabelle 4.1 nach der Chronologie ihres Auftretens geordnet und haben folgenden zweistufigen Aufbau: [Routing-Layer][HighLevel-Layer]. An den Anfang der Nachricht setzt der Rotuing-Layer eine acht Bit lange, eindeutige Message-ID, die dazu dient, Broadcats-Nachrichten nur einmal zu verschicken, da Duplikate anhand der gleichen ID erkennbar sind. Danach folgt ein Bit (BC Bit), welches angibt, ob es sich bei der Nachricht um eine Broadcast-Nachricht handelt. Nachrichten, die auf jeden Fall als Broadcast verschickt werden sind: Init-Message, Register-All und Factoryelement. Nach dem Broadcast-Bit steht die Adresse der Nachricht, die sich aus zwei Koordinaten mit je sieben Bit Größe zusammensetzt.

Die Informationen, die der HighLevel-Layer den Nachrichten beifügt, sind in einem variablen Format kodiert, um möglichst geringe Nachrichtengrößen zu erhalten. Zu Beginn befindet sich der Nachrichtentyp, der eine Breite von sechs Bit braucht. Abhängig vom Typ, werden die dann folgenden Daten der Nachricht interpretiert. Zum Beispiel werden bei der Init-Message die 21 Bit nach dem Typ so ausgewertet, das die ersten 14 Bit

als Adressse interpretiert werden, dann folgen die beiden als Orientierungsbits und die letzten 5 Bit werden als HopCounter interpretiert.

Nachrichtenname	Routinglayer		HighLevel		
	BC (1Bit)	Adresse (14Bit) (x,y)	Type (6Bit)	Data Größe d. Felder	Beschreibung
Init-Message	1	-	0	[14, 1, 1,5]	Adresse x, X-hochzählen, Y-hochzählen, Hop-Counter
Init-Response	0	Parkplatz	1	[14]	Hostadresse
Register-All	1	-	2	[56]	4 Hostadressen (erste Adresse ist Adresse des Master)
Register-Response	0	Master	3	[0]	-
Register-Checkpoint	0	Master	4	[14]	Feldkoordinaten
Send-Checkpointnumber	0	Feld	5	[3]	Checkpointnummer (1-8)
Send-Checkpoints	0	Alle Clients	6	[112]	Checkpointadressen
Cards	0	Alle Clients	7	[70]	10 Karten á 7Bit
Programsequence	0	Master	8	[35]	5 Kartenprioritäten á 7Bit
Next-Order-Message	0	Client	9	[0]	-
Order	0	Feld	10	[4,8, 3]	Bewegung, Bitvektor für aktive Optionskarten und Runde x von 5
Order-Response	0	Client	11	[14,2,4,2,3,1]	Roboterposition, Blickrichtung, Schaden, Repairfeld (nein, einfach, zweifach), Checkpointnummer, Dead-Flag
Next-Order-Response	0	Master	12	[0]	-
Factoryelement	1	-	13	[2,3]	Factoryelement-Typ, Programmschritt (x von 5)

	BC (1Bit)	Adresse (14Bit) (x,y)	Type (6Bit)	Data	
				Größe d. Felder	Beschreibung
Factoryelement-Response	0	Client	14	[14,2,4,2,3,1]	Roboterposition, Blickrichtung, Schaden, Repairfeld (nein, einfach, zweifach), Checkpointnummer, Dead-Flag
Statusupdate	0	alle Clients	15	[2,1,4,2,3,17,3]	Roboter-ID Gewinn-Flag, Schaden, Leben, Checkpoint, Optionskarten, Programmschritt(x von 5)
Factoryelement-Done	0	Master	16	[0]	-
EndofRound	0	Clients	17	[0]	-
Request-Option	0	Master	18	[2]	ClientID
Send-Option	0	Client	19	[17]	Optionskarte
Return-Cards	0	Master	20	[70,17]	10 Karten á 7Bit, alle Optionskarten
Vote-Message	0	Master	21	[1]	Ja, Nein
Reset-Message	0	Alle Client	22	[1]	Ja, Nein
Reset-Ready	0	Alle Client	23	[0]	-
Leaving-Message	0	Alle Client	24	[2]	ClientID
RobotDead-Message	0	Feld	25	[0]	-
Toggle-Laser	0/1	Feld-Feld / Master-Feld	26	[1,2]	An/Aus-Flag, CountUpXY
Query	0	Nachbar-Feld	27	[14,1,1,3,1]	Antwortadresse, Verschiebe-Bit, Schadenverursachend (Ja/Nein), Checkpointnumber, Dead-Bit
Query-Response	0	Nachbar-Feld	28	[1,14]	Ja, Nein, Angefragtes Feld
Laser	0	Nachbar-Feld	29	[2, 2, 2, 1,1]	Orientierung (Laserrichtung), Stärke, Leistungstoken, Traktorstrahl, Hochdruckstrahl

	BC (1Bit)	Adresse (14Bit) (x,y)	Type (6Bit)	Data	
				Größe d. Felder	Beschreibung
Robot-State	0	Nachbar-Feld	30	[2,2,8,1,3, 4,4,14,4,4]	RoboterID, Blickrichtung, Optionskarten, Life-Bit, Runde, Fabrikelemente-stage, Nicht-Gemeldeter Schaden, Vorherige Position, Nicht-Gemachte-Drehungen, Nicht-Gemachte Bewegung

Tabelle 4.1: Nachrichtenspezifikation

4.3.3.2 Zustandsübergangsmodell des Application-Layers

Die Aufgaben des Application-Layers lassen sich am besten als Zustandsübergangsmodell modellieren. Dieses Modell beinhaltet eine Menge von Zuständen zwischen denen das System wechseln kann. Diese Zustandsübergänge sind aber an gewisse Bedingungen geknüpft, wie zum Beispiel die Ankunft einer Nachricht. Das komplette Zustandsübergangsmodell wird in Abbildung 4.3 dargestellt.

Die Zustände in der Abbildung sind als Viereckige dargestellt, die Zustandsübergänge durch Pfeile. Ist an einen Zustandsübergang eine Bedingung geknüpft, so wird sie in eckigen Klammern an den Pfeil geschrieben (vgl. Übergang von B_Init nach Normal). Findet zusätzlich zum Übergang noch eine Aktion statt, so steht sie nach einem Forward-Slash (vgl. Übergang von Orderprocessing nach Normal). Ist an den Zustandsübergang sowohl eine Bedingung als auch eine Aktion genüpft, wird die wie folgt notiert: [Bedingung]/Aktion.

Nach dem Einschalten gelangt das System zuerst in einen initialen Zustand, wobei die Spielfelder in den Zustand **B_Init** kommen, während der Parkplatz in **P_Init** startet. In diesem Zustand werden auf der Ebene des Routing-Layers die *Init-Message*s bearbeitet. Empfängt ein Host-PC eine *Init-Message*, sendet er seine Adresse mit einer *Init-Response* an den Parkplatz zurück. Der Parkplatz sammelt die Responses und broadcastet diese mit einer *Register-All* an alle Felder und Hosts. Dies bewirkt im Spielfeld und im Parkplatz jeweils einen Zustandsübergang nach **Normal**.

Im Zustand **Normal** werden Querys mit einer *Query-Response* bearbeitet, einen Zustandsübergang löst die Nachricht *Order* aus. Das System wechselt in den Zustand **Orderprocessing**, in dem der Befehl ausgeführt wird.

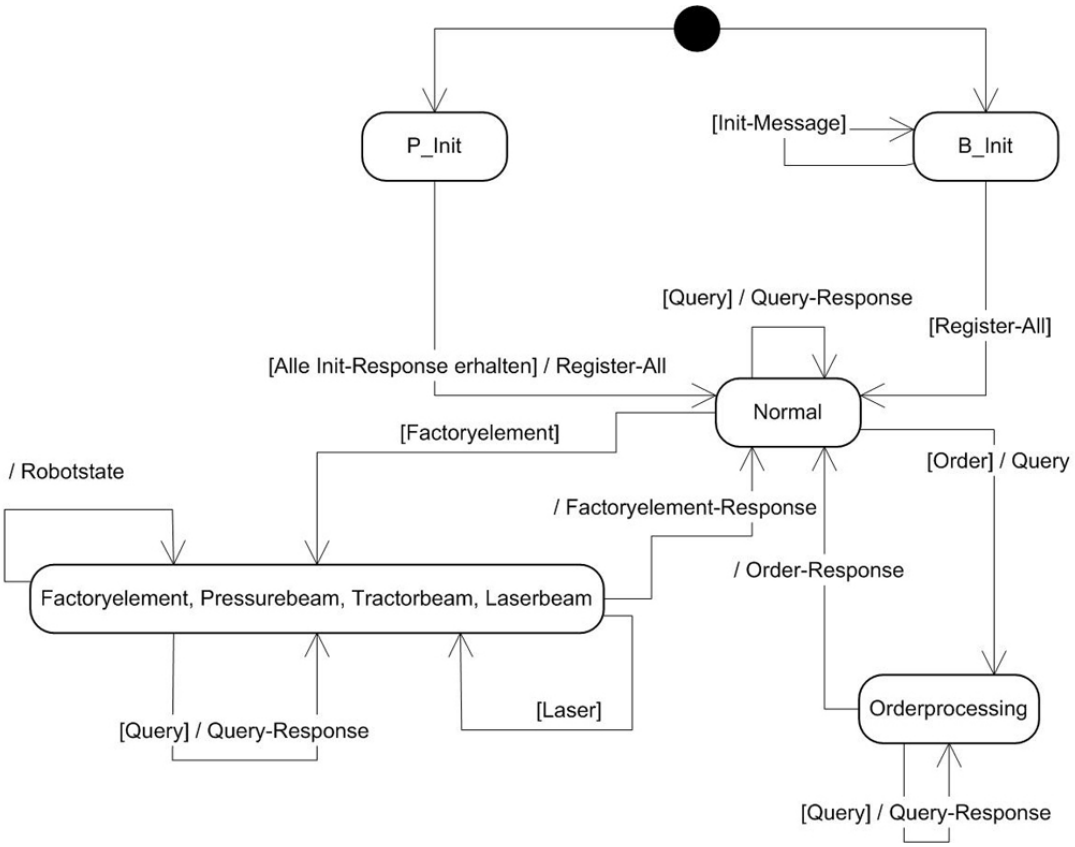


Abbildung 4.3: Statemachine

Die Befehlsausführung endet mit dem Senden der *Order-Response* Nachricht, die dem Client alle neuen Daten über seinen Roboter mitteilt.

Ähnlich zur Befehlsausführung läuft die Abarbeitung der Fabrikelemente ab. Eine *Factory-Element*-Nachricht führt das System in den Zustand **Factoryelement, Pressurebeam, Tractorbeam, Laserbeam**. Dieser Zustand besteht aus mehreren Unterzuständen, die aber zur Vereinfachung des Diagramms zusammengefasst wurden.

Der **Factoryelement** Zustand führt zur Abarbeitung der Fabrikelemente Expressförderband, Förderband, Schieber, Zahnrad, Presse. Danach folgen die Zustände **Pressurebeam** und **Tractorbeam**, in denen diese beiden Optionskarten aktiviert werden. Das ist nötig, weil dadurch Roboter verschoben werden können, und somit nur dann ein deterministisches Verhalten garantiert werden kann, wenn auch diese Optionskarten in eine totale Ordnung mit den anderen Fabriklelementen gebracht werden. Als letztes Fabrikelement treten die Laser der Roboter und der Felder in Kraft und fügen den Robotern Schaden zu.

4.3.3.3 Implementierung

Zur Implementierung des Zustandsübergangsmodell wird ein Array mit Funktionspointern verwendet. Das Array hat 2 Dimensionen eine für den Nachrichtentyp und eine für den aktuellen Zustand. Beim Eintreffen einer Nachricht wird einfach die Funktion an Position [aktueller Zustand][Nachrichtentyp] aufgerufen. Für jeden Nachrichtentyp, den das Spielfeld verarbeiten muss, gibt es eine Funktion. Wenn in einem bestimmten Zustand ein bestimmter Nachrichtentyp bearbeitet werden soll, wird die korrespondierende Funktion aufgerufen.

Dadurch wird der Code sehr schlank und ist sehr wartungsfreundlich. Im folgenden Teil werden jetzt kurz die Funktionen für jeden Nachrichtentyp, den das Sielfeld bearbeiten muss, beschrieben:

HandleInit Wird bei eintreffen einer *Init-Message* aufgerufen. *Init-Message*s werden nur im **Init**-Zustand bearbeitet. Die Funktion berechnet die Koordinaten aller ihrer neuen Felder und initialisiert weitere Datenstrukturen für die Orientierung.

RegisterClients Die *Register-Client*-Nachricht enthält die Adressen aller Clients und die des Masters, diese werden abgespeichert, um die Response-Nachrichten an die zugehörigen Konsolen der Roboter zu schicken. Die Funktion witrd nur im Zustand **B_Init** aufgerufen.

HandleQuery Querys werden immer dann gesendet, wenn ein Roboter ein Feld betreten will, oder wenn ein Roboter einen mechanischen Arm hat und dadurch festgestellt werden muss, ob das Feld einen Checkpoint besitzt. Ein sog. forcedisplace-Flag gibt an, ob der Roboter, der das Feld betreten will in der Lage ist einen anderen zu schieben oder zum Beispiel einen Rammbock hat und dadurch Schaden zufügt. HandleQuery wird in allen Zuständen ausser im Init-Zustand aufgerufen.

HandleQueryResponse wird in allen Zuständen ausser im Init-Zustand aufgerufen. Gesendet wird eine solche Nachricht nach einem Query (also als Antwort. ob der Roboter das Feld betreten darf). Zuerst wird festgestellt, zu welchem Query diese Response gehört, um abhängig davon weitere Schritte einzuleiten.

HandleOrder Wird aufgerufen, wenn der Roboter einen Befehl ausführen soll. Dreh-Befehle können sofort zum Roboter geschickt und ausgeführt werden, während Bewegungsbefehle zuerst zu einem Query führen. Der Befehl kann erst ausgeführt werden, wenn klar ist, dass das Zielfeld existiert und frei ist. Die Ausführung des Befehls findet erst statt, wenn die QueryRespnse eingetroffen ist.

HandleFactoryElement Wird nur im Zustand Normal aufgerufen. Es werden nacheinander die Fabrikelemente Expreßförderband, Förderband, Schieber, Zahnrad, Presse ausgeführt. Im Roboterstate wird vermerkt, welche Stufen er bereits abgearbeitet hat, so dass keine mehrfach durchlaufen wird.

HandlePressureBeam Wird nur im Zustand Factoryelement ausgeführt. Hat ein Roboter die Optionskarte Hochdruckstrahl, so wird dieser jetzt eingesetzt.

HandleTractorBeam Analog zu HandlePressureBeam.

HandleLaserBeam Durchläuft alle Felder und prüft, ob ein Roboter darauf steht oder das Feld einen Laser hat. Daraufhin wird eine Laser-Nachricht los geschickt, so dass alle weiteren Felder in Schussrichtung entsprechend Schadenspunkte verteilen.

HandleLaser Wird aufgerufen, wenn eine Laser Nachricht eintrifft. Die Nachricht wird an das nächste Feld in Schussrichtung weitergeleitet, außer es ist ein Roboter oder eine Wand ist dazwischen. Die Nachricht hat zusätzlich ein Flag, das angibt, ob es sich um einen Traktorstrahl oder Hochdruckstrahl handelt, in dem Fall wird ein getroffener Roboter verschoben.

SaveRobotState Wird aufgerufen, wenn dem Feld ein Roboterstatus zugeschickt wird. Der Roboterzustand enthält zahlreiche Informationen unter anderem durchlaufene Fabrikelemente und noch nicht erledigte Dreh-/Bewegungsbefehle.

SaveCheckpointNumber Teilt dem Spielfeld mit, welche Nummer der vom Benutzer eingesteckte Checkpoint hat.

ToggleLaser Invertiert den Zustand der LEDs, die die Laserstrahlen visualisieren. Die Laser sollen während der Spieler seinen Roboter programmiert und während die Fabrikelemente dargestellt werden.

HandleEndOfRound Macht die Laser LEDs an, um Laser darzustellen und so das Programmieren der Roboter zu erleichtern.

In obiger Aufzählung sind nur die High-Level Funktionen enthalten, die eine Reihe weiterer Hilfsfunktionen aufrufen. Diese Hilfsfunktionen dienen zur Generierung von Antwortnachrichten, Senden von Nachrichten an Roboter und dem Rechnen mit Koordinaten etc.

Die Implementierung des High-Level Layers ist abgeschlossen und es wurde auch bereits mit den ersten Test begonnen. Im nächsten Semester stehen weitere Test in der Testumgebung des Routing-Layers (vgl. 4.3.2.5) auf dem Plan, bevor umfangreiche Integrationstests durchgeführt werden.

4.3.4 Scheduler

Oberhalb des Protokollstack (siehe 4.1) ist ein Scheduler angesiedelt. Obwohl diese Softwarekomponente nicht direkt zum Protokoll-Stack gehört, ist sie dennoch das Bindeglied zwischen den Schichten des Stacks. Jegliche Kommunikation zwischen den Schichten wird duch das Erzeugen von Jobs durchgeführt, die dann in den Scheduler eingetragen und nacheinander abgearbeitet werden.

Während es für Protokolle für PCs nicht ungewöhlich ist, dass jede Schicht jeweils einen eigenen Thread verwendet, um parallel zu arbeiten, kann diese Technik nicht direkt in unseren Komponenten eingesetzt werden. Um trotzdem parallel arbeiten zu können wird der beschriebene Scheduler eingesetzt. Die Schichten des Stacks arbeiten so zumindest scheinbar parallel. Die Jobs werden von den jeweiligen Komponenten des Stacks erzeugt und dann vom Scheduler auf *"first come, first serve"*-Basis abgearbeitet. Durch Interrupt-Routinen des Mikrocontrollers kann dieser Prozess unterbrochen werden, um neue Nachrichten in die Warteschlange des Schedulers einzuhängen.

4.4 Host-Applikation

4.4.1 Bedienkonzept

Die Aufgabenstellung sieht gemäß PG-Antrag vor, Software für einen Host-PC zu entwickeln, mit der die Spieler ihre Eingaben an die Roboter übermitteln können. Im Unterschied zur Variante, die im PG-Antrag vorgeschalgen wird, wurde beschlossen, die Software so zu konzipieren, dass nicht alle Spieler vor einem PC sitzen und ihre "Programmierung" vornehmen, sondern pro Spieler ein PC angeschlossen wird. Die Spieler können so ihre Eingaben parallel vornehmen und sind dadurch keinen Wartezeiten aufgrund der Eingaben ihrer Mitspieler ausgesetzt. Zusätzlich wird so auch vermieden, dass die Spieler die Karten ihrer Mitspieler einsehen können. Dadurch wird es jedoch notwendig, einen der Host-PCs intern als Master auszuzeichnen. Der Master-PC wird benötigt um die Konsistenz der Karten aufrecht zu erhalten und für die Abarbeitung der Programmbefehle der Spieler gemäß ihrer Kartenprioritäten zu sorgen.

4.4.2 wxWidgets

Die Host-Software wird in C/C++ geschrieben. So kann der in reinem C geschriebene Quelltext der Kommunikations-Architektur auch in den Hosts wiederverwendet werden. In C/C++ stehen jedoch nicht wie in Java durch die *Swing*-Pakete eigene Klassen zum erstellen grafischer Benutzeroberflächen zur Verfügung. Es wurde daher entschieden *wxWidgets*[2] einzusetzen, eine Open-Source Bibliothek zur Generierung von GUIs. Die Vorteile von *wxWidgets* gegenüber anderen GUI-Bibliotheken liegen vor allem in der Plattformunabhängigkeit und der plattformnativen Darstellung. Die plattformnativität wird dadurch erreicht, dass *wxWidgets* intern plattformspezifischen Code einsetzt. Daher müssen die GUI-Bibliotheken von *wxWidgets* zunächst für die Zielplattform selbst compiliert werden. Außerdem unterliegt *wxWidgets* im wesentlich der L-GPL (Library General Public Licence). Die L-GPL entspricht der GPL mit dem Zusatz, dass der kompilierte Quelltext frei eingesetzt werden kann und nicht offen gelegt oder ebenfalls als Open-Source Projekt verfügbar gemacht werden muss.

[2] http://wxwindows.org/

Details können der Referenz [Sma05] entnommen werden. Die Referenz umfasst Installationshinweise, Hilfe zum Einrichten der IDE für die Verwendung mit *wxWidgets* und Erläuterungen und Code-Beispiele zu den *wxWidgets*-Klassen.

4.4.3 Programmablauf

Der Ablauf des Programms kann im wesentlichen in drei Phasen eingeteilt werden:

- Eine Initialisierungsphase, in der die Software startet und sich konfiguriert,
- die Spielphase, in der die Spieler ihre Eingaben vornehmen und
- die Abschlussphase, in der die Spielauswertung erfolgt.

4.4.3.1 Initialisierungsphase

Während das Spiel startet, findet im Feld eine Initialisierung statt. Dabei wird im Spielfeld ausgehend vom "Parkplatz" ein Koordinatensystem aufgebaut, so dass die einzelnen Spielfeld-Boxen direkt adressiert werden können. Dasselbe gilt auch für die angeschlossenen Host-PCs. Diese werden von der Routing-Schicht wie ein weiterer Spielfeld-Boxen interpretiert. Hinzu kommt jedoch, dass sich die Hosts beim Parkplatz zurückmelden. Der erste Host, der sich beim Parkplatz zurückmeldet, wird als Master ausgezeichnet. Dieser Host-PC übernimmt die oben beschriebenen notwendigen zentralen Aufgaben.
Sobald sich alle vier Host-PCs zurückmelden oder ein Timer abläuft, wird der Spieler, der den Master stellt, dazu aufgefordert, in aufsteigender Reihenfolge, nacheinander die Checkpoints auf dem Spielfeld zu verteilen. Sobald der Spieler bestätigt, das er alle Checkpoints platziert hat, ist die Initialisierungsphase abgeschlossen. Läuft der Timer ab bevor sich alle vier Host-PCs zurückgemeldet haben wird die Spielerzahl auf die Zahl der bis dahin zurückgemeldeten PCs gesetzt. Wenn sich kein oder nur ein Spieler zurückmeldet, wird der Timer neu gestartet.

4.4.3.2 Spielphase

Sobald die Initialisierungsphase abgeschlossen ist, startet die Spielphase. Den Spielern wird eine Programmoberfläche präsentiert, die alle wichtigen Informationen für den nächsten Zug anzeigt:

- Die zur Verfügung stehenden Programmierkarten,
- die Programmslots für die Karten, die im nächsten Zug verwendet werden,
- den Status, ob und welche Karten bereits festgebrannt sind,

- den Button um den Roboter für die nächste Runde abzuschalten und
- Informationen über:
  - Erlittene Schadenspunkte,
  - verbliebene Backups,
  - den nächsten zu erreichenden Checkpoint und
  - welche Optionskarten im Besitz des Spielers sind.

Die Spieler können damit ihren nächsten Zug planen und so ihre Befehlssequenz zusammenstellen. Sobald die Spieler ihren Zug abgeschlossen und bestätigt haben, schicken die Host-PCs die Kartenprioritäten an den Master-PC, damit dieser für die Befehlsabarbeitung in der korrekte Reihenfolge sorgen kann. Der Master-PC wird die Client-PCs der Reihenfolge nach notifizieren, dass diese nun ihren Befehl an das Spielfeld übermitteln können.
Während die Host-PCs die Befehle der Spieler an das Feld übertragen, wird den Spielern eine Übersicht präsentiert, der die Spieler entnehmen können, wie weit die Befehlsabarbeitung fortgeschritten ist und wie der Status der andern Roboter aussieht. Diese Informationen werden laufend durch das Spielfeld aktualisiert, so dass die Spieler nachvollziehen können, wodurch beispielsweise Schadenspunkte zustande kommen.

4.4.3.3 Abschlußphase

Nachdem ein Spieler mit seinem Roboter alle Checkpoints in der richtigen Reihenfolge passiert hat, endet das Spiel. Die Software fragt die Spieler, ob das Spiel in gleicher Zusammensetzung noch einmal gespielt werden soll. Bestätigen alle Spieler diese Frage, werden die Roboter zurückgesetzt und das Spiel beginnt von neuem.
Lehnt jedoch nur ein Spieler ab, das Spiel zu wiederholen, werden alle Spieler benachrichtigt, dass das Spiel neu aufgebaut werden muss, um eine weitere Partie zu spielen. Die Software kann nun entweder beendet werden, oder sie kehrt in die Initialisierungsphase zurück und wartet darauf, dass das Spielfeld neu aufgebaut und wieder angeschaltet wird.

4.4.4 GUI-Elemente

4.4.4.1 Splashscreen

Der Splashscreen (Abb. 4.4) wird während der Initialisierungsphase angezeigt. Im Verlauf des Projekts wird dieser Splashscreen noch um die Möglichkeit erweitert, anzuzeigen, in welchem Teil der Initialisierungsphase sich das Spiel grade befindet. Der Benutzer erhält so nicht den Eindruck, dass das Programm nicht mehr reagiert und abgestützt ist.

Abbildung 4.4: Splashscreen

4.4.4.2 Hauptbildschirm

Der Hauptbildschirm (Abb. 4.5) zeigt dem Spieler unter dem Logo seine Spielkarten an. Die Karten werden, von links nach rechts, mit fallender Priorität angeordnet, um dem Spieler eine bessere Übersicht zu ermöglichen.

Auf der linken Seite ist der Bereich mit den Kartenslots angeordnet. Oberhalb der Slots wird durch Warndreiecke angezeigt, ob die Karte, und damit der Slot, aufgrund von Schadenspunkten gesperrt ist. Unterhalb der Slots befindet sich der Button, mit dem der Spieler seine programmierte Befehlssequenz bestätigen und an den Master senden kann. Dieser Button ist jedoch deaktiviert, bis der Spieler alle Slots mit Karten gefüllt hat.

Auf der rechten Seite werden dem Spieler die Informationen präsentiert, die im Abschnitt 4.4.3.2 auf Seite 167 beschrieben sind. Zusätzlich können hier die erhaltenen Optionskarten auch für die kommende Runde an- und abgeschaltet werden.

Am unteren Bildschirmrand ist noch der Exit-Button angeordnet, mit dem der Spieler jederzeit zwischen den Runden das Spiel verlassen kann.

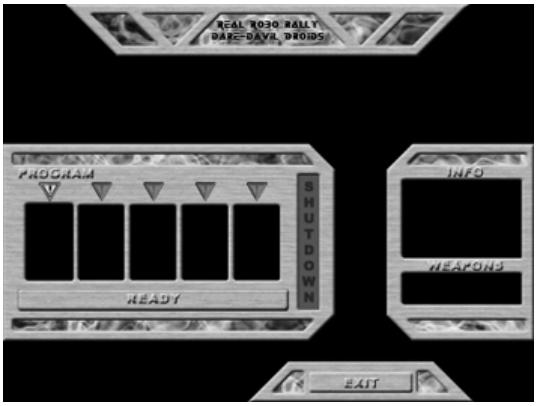


Abbildung 4.5: Hauptbildschirm

4.4.4.3 Übersichtsbildschirm

Der Übersichtsbildschirm (Abb. 4.6) wird dem Spieler präsentiert, sobald er auf dem Hauptbildschirm seine Befehlssequenz bestätigt hat. Die Felder enthalten die gleichen Informationen, die im Infobereich des Hauptbildschirms angezeigt werden. Die Spieler können hier allerdings auch die Informationen über die Roboter ihrer Mitspieler einsehen.

Im unteren Bereich der Übersicht ist ein Fortschrittsbalken eingelassen. Die Spieler können so erkennen, wie weit die Befehlsabarbeitung der Roboter im Spielfeld fortgeschritten ist. Die Felder des Fortschrittsbalkens sind rot, wenn noch keine Befehle bearbeitet worden sind. Sobald ein Befehl von allen Robotern ausgeführt wurde, färbt sich das zugehörige Fortschrittsfeld gelb. Nachdem alle Befehle ausgeführt wurden, färbt sich der gesamte Balken grün, um dem Spieler zu signalisieren, dass die nächste Runde beginnen kann. Durch Betätigen des Continue-Buttons kehrt der Spieler zum Hauptbildschirm zurück.

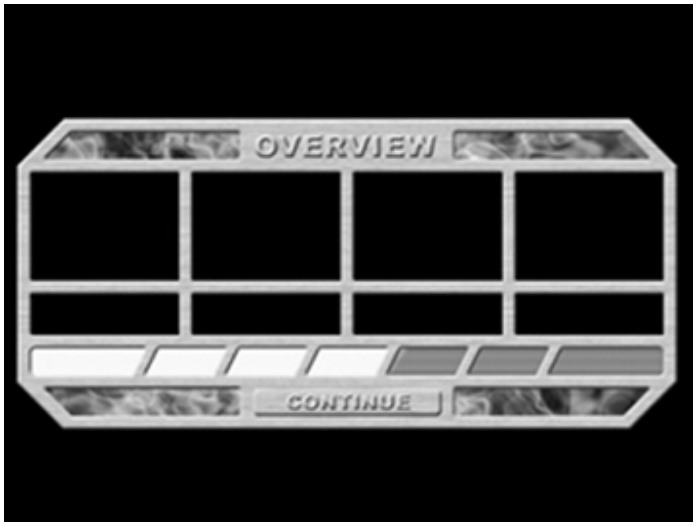


Abbildung 4.6: Übersichtsbildschirm

4.4.4.4 Karten

Die Karten (Abb. 4.7) mit den Bewegungsbefehlen, die dem Spieler zum Programmieren seines Roboters angezeigt werden, sind der amerikanischen Variante des Original-Spiels entliehen. In das Feld in der oberen rechten Ecke der Karten wird vom Spiel die Priorität der Karte eingetragen, mit der der entsprechende Bewegungsbefehl vom Spiel ausgeführt wird.

Abbildung 4.7: Karten

4.4.4.5 Kombinierte Karten

Durch die Optionskarten “Dual Processor“ und “Crab-Legs“ wird es dem Spieler ermöglicht, die Befehlskarten “Move 1“ und “Rotate Left/Right“ zu kombinieren und in einem Programmslot abzulegen. Diese Befehle werden daraufhin in einem Schritt ausgeführt. Diese Karten werden im Original-Spiel dadurch dargestellt, dass der Spieler die beiden zu kombinierenden Karten im gleichen Slot ablegt.

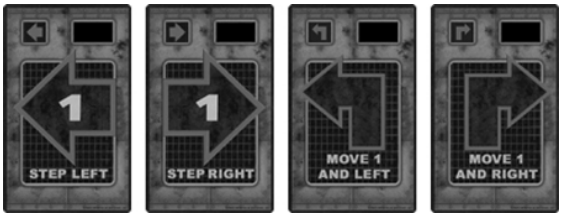


Abbildung 4.8: Kombinierte Karten

Es wurde jedoch beschlossen diese Kombinationen als einzelne Karten (Abb. 4.8) zu ergänzen, damit der Spieler auf einfache Weise erkennen kann, welche Karten in einem Slot kombiniert sind. Der kombinierten Karte wird dabei die Priorität des “Move 1“-Befehls zugeordnet.

4.4.4.6 Optionskarten

Die Optionskarten werden durch kleine Symbole (Abb. 4.9) im unteren Bereich der Informationsleiste dargestellt. Die Symbole, die für die Darstellung der Optionskarten verwendet werden, sind von den Optionskarten aus dem Original-Spiel übernommen. Wenn der Spieler die Maus über das Symbol der Optionskarte bewegt, wird ein Tooltip angezeigt, in dem die Funktion der Karte erklärt wird.

Wird das Symbol der Optionskarte angeklickt, werden die Konfigurationsmöglichkeiten der Karte im oberen Bereich der Informationsleiste angezeigt. Je nach Funktion kann die Karte hier an- bzw. abgeschaltet werden, oder es wird angezeigt, wie oft die entsprechende Zusatzfunktion noch eingesetzt werden kann.

Abbildung 4.9: Optionskarten

5 Zusammenfassung

5.1 Status

Der Status-Abschnitt soll als kurze Zusammenfassung der vorhergehenden Kapitel dienen, um einen groben Überblick der Fortschritte der Projetkgruppe R3D3 zu bekommen. Die Struktur des Abschnitts orientiert sich an dem Aufbau der zusammengefassten Kapitel.

5.1.1 Hardware

5.1.1.1 Roboter

Da für den Roboter keine Fertiglösung gewählt werden konnte, mussten aus einer Vielzahl von Bauteilen die richtigen ausgewählt werden. Als besonders aufwändig erwies sich die Suche und der Zusammenbau der Getriebe und das Auffinden eines geeigneten Akkus. Diess ist auf die besonderen Anforderungen bezüglich der Größe zurückzuführen. Fast alle Komponenten müssen zuerst auf ihre Größe bezüglich der Verwendbarkeit geprüft werden.

Um alle elektrischen Bauteile unterzubringen, werden im Roboter mehrere Platinen eingesetzt. Schließlich wurden alle benötigten Bauteile bestellt, getestet und zu einem Prototypen zusammengesetzt werden. Dies ist in diesem Semester erfolgreich abgeschlossen worden, so dass bereits damit begonnen wurde, die Software für den Roboter auf dem Prototypen zu testen.

5.1.1.2 Spielfeld

Ähnlich wie bei den Robotern gab es auch für das Spielfeld keine Fertiglösung, die alle Anforderungen zufriedenstellend erfüllt hätte. Daher musste zuerst eine Entscheidung getroffen werden, welche Materialien verbaut werden. So sind die Seitenwände aus Epoxydharz-Platinen, die mit 4 Aluminium-Winkeln verschraubt werden. Eine durchsichtige Makrolon-Platte bildet die Oberfläche. Die bestellten Materialien reichen aus, 16 Spielfeldboxen und einen Parkplatz zu fertigen, im Moment ist eine Box als Testbox einsatzbereit.

Da sowohl zur Navigation als auch zur Kommunikation des Roboters LEDs benutzt werden, wird das Design der im Spielfeld enthaltenen Elektronik stark davon geprägt,

eine Vielzahl LEDs anzusteuern. Eine Möglichkeit bestand darin, die Ansteuerung mit Hilfe von Latches zu realisieren. Dieser Ansatz wurde aber wieder verworfen, weil er zu teuer war und zu viel Platz verbrauchte.

Ein weiterer Ansatz verwendet einen CPLD, so dass alle Bauteile auf eine einzige Platine passen, was auch den mechanischen Aufbau vereinfacht. Ein Prototyp dieser Platine ist fertig gestellt und kann nun getestet werden.

Der Entwurf für die Schnittstelle zwischen Host-PC und Spielfeld ist fertig und ein einzelnes Exemplar ist bereits hergestellt.

5.1.2 Software

5.1.2.1 Protokoll-Stack

Der Protokoll-Stack ist vollständig implementiert und in einer künstlichen Testumgebung erfolgreich getestet worden. Die Kommunikation der Spielfeldbausteine wurde in der Testumgebung dadurch simuliert, dass die Low-Level Schicht durch eine Implementierung auf Socket-Basis ersetzt wurde.

Das Protokoll zur Low-Level Kommunikation der Spielfelder untereinander und mit dem Roboter ist ebenfalls implementiert und einsatzfähig.

5.1.2.2 Host-Applikation

Der Schwerpunkt der Aufgaben für dieses Semester lag vor allem bei der Kommunikationsprotokoll-Entwicklung und dem Design von Hardware, so dass im Bereich des Host nur kleine Fortschritte gemacht wurden. So ist die GUI-Schicht der Host-Architektur, bis auf den Abschlussbildschirm, unter Verwendung der *wxWidgets*-Bibliotheken, fertig implementiert worden. Es fehlt jedoch noch die Anbindung an die unterliegende Kommunikations-Architektur und die Applikationslogik für den Spielablauf.

5.2 Ausblick

5.2.1 Hardware

5.2.1.1 Roboter

Das Team B, dass sich mit dem Fertigen der Roboter beschäftigt, muss im nächsten Semester vier Roboter herstellen. Dabei kann der in diesem Semester gefertigte Roboter mitbenutzt werden. Für alle der vier Roboter muss noch ein Gehäuse entwickelt werden, das sowohl optischen als auch funktionalen Ansprüchen genügt. Vorher wird in einem

kurzen Re-Design-Prozess das Platinen-Layout überarbeitet, da es dort während der Tests einige Probleme gab.

5.2.1.2 Spielfeld

Bei den Spielfeldern ist vor allem die "Serienproduktion" im nächsten Semester zu leisten. Gemäß Abschnitt 1.2 sind bis zum Ende der Projektgruppe 16 Spielfelder und zusätzlich ein Parkplatz zu fertigen. Das bedeutet, dass der mechanische Aufbau bestehend aus Winkel, Seitenwänden und Makrolonplatte im 17 facher Ausführung bereit stehen muss. Das Material für vier Boxen ist bereits fertig.

Auf der Seite der Elektronik werden im nächsten Semester die nötigen Platinen geätzt und bestückt, so dass sie in die Spielfelder eingebaut werden können. Die Schnittstelle zwischen Host und Spielfeld ist konzipiert und es ist bereits ein Exemplar erstellt worden, so dass im nächsten Semester noch drei weitere zum Anschluss von insgesamt vier Host-PCs gefertigt werden müssen.

5.2.2 Software

5.2.2.1 Protokoll-Stack und Host-Applikation

Bei der Software geht es im kommenden Semester vor allem um die Integration von Hardware und High-Level Software. Das bedeutet, dass der in Abschnitt 4.3 beschriebene Protokollstack auf dem Mikrocontroller zum Einsatz kommt. Dabei werden umfangreiche Integrationstests durchgeführt, um die spezifizierten Schnittstellen ausgiebig zu testen.

Die bestehenden Konzepte für den Host-PC werden nach einer kurzen Überarbeitungsphase implementiert und in einem umfassenden übergreifenden Integrationstest überprüft, sobald die Hardware-Teams eine geeignete Menge an Komponenten bereitstellen können.

6 Anhang

6.1 Schaltpläne

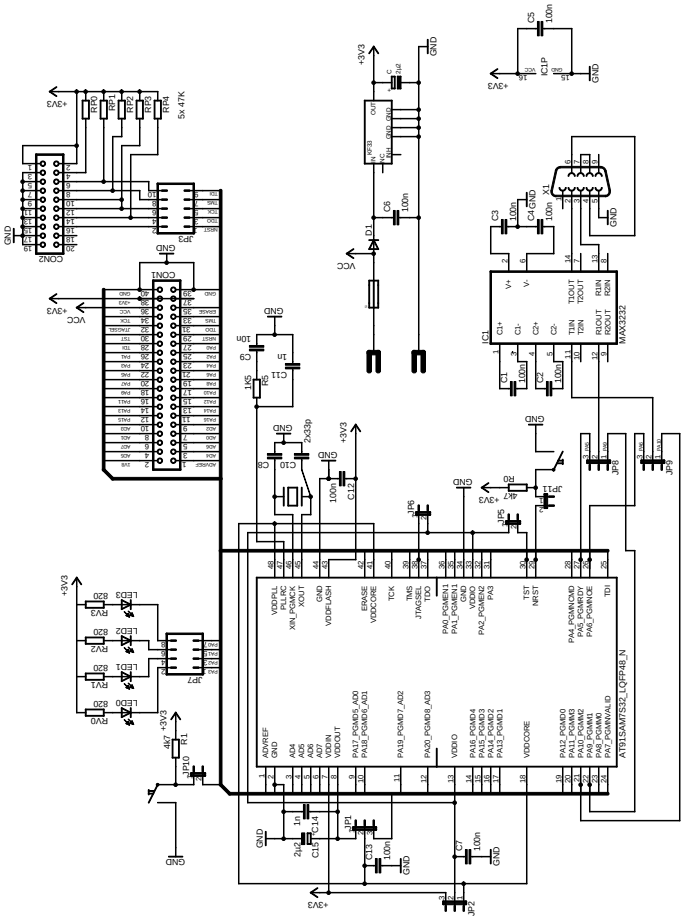


Abbildung 6.1: Schaltplan Testboard AT91SAM7S32

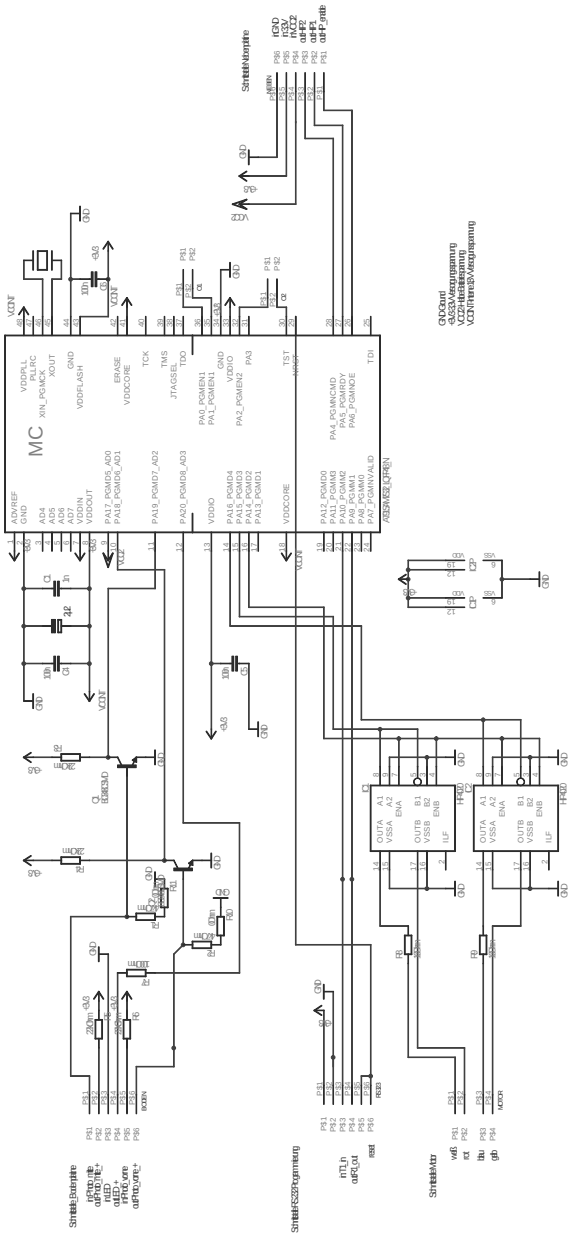


Abbildung 6.2: Schaltplan Roboter Hauptplatine

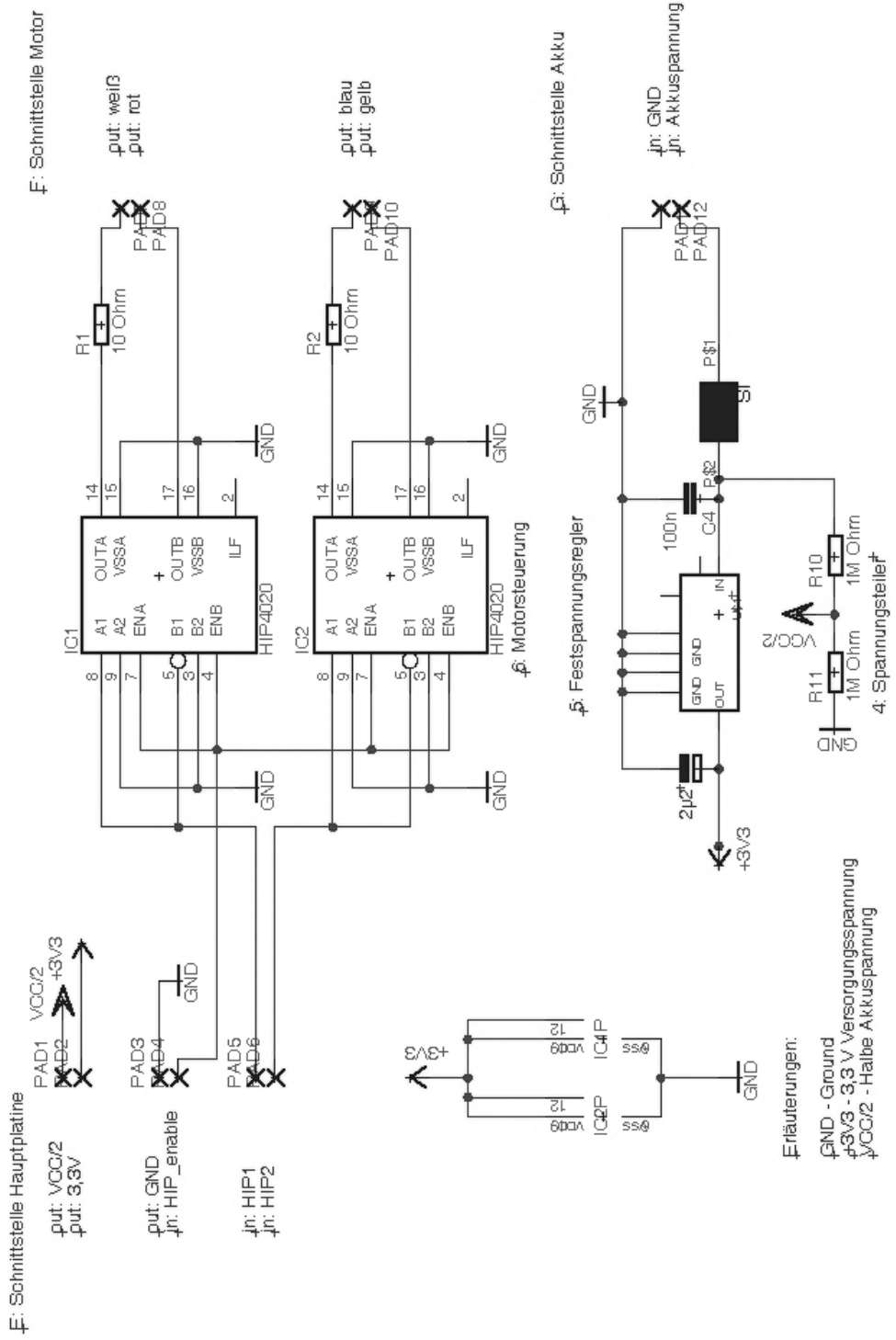


Abbildung 6.3: Schaltplan Roboter Nebenplatine

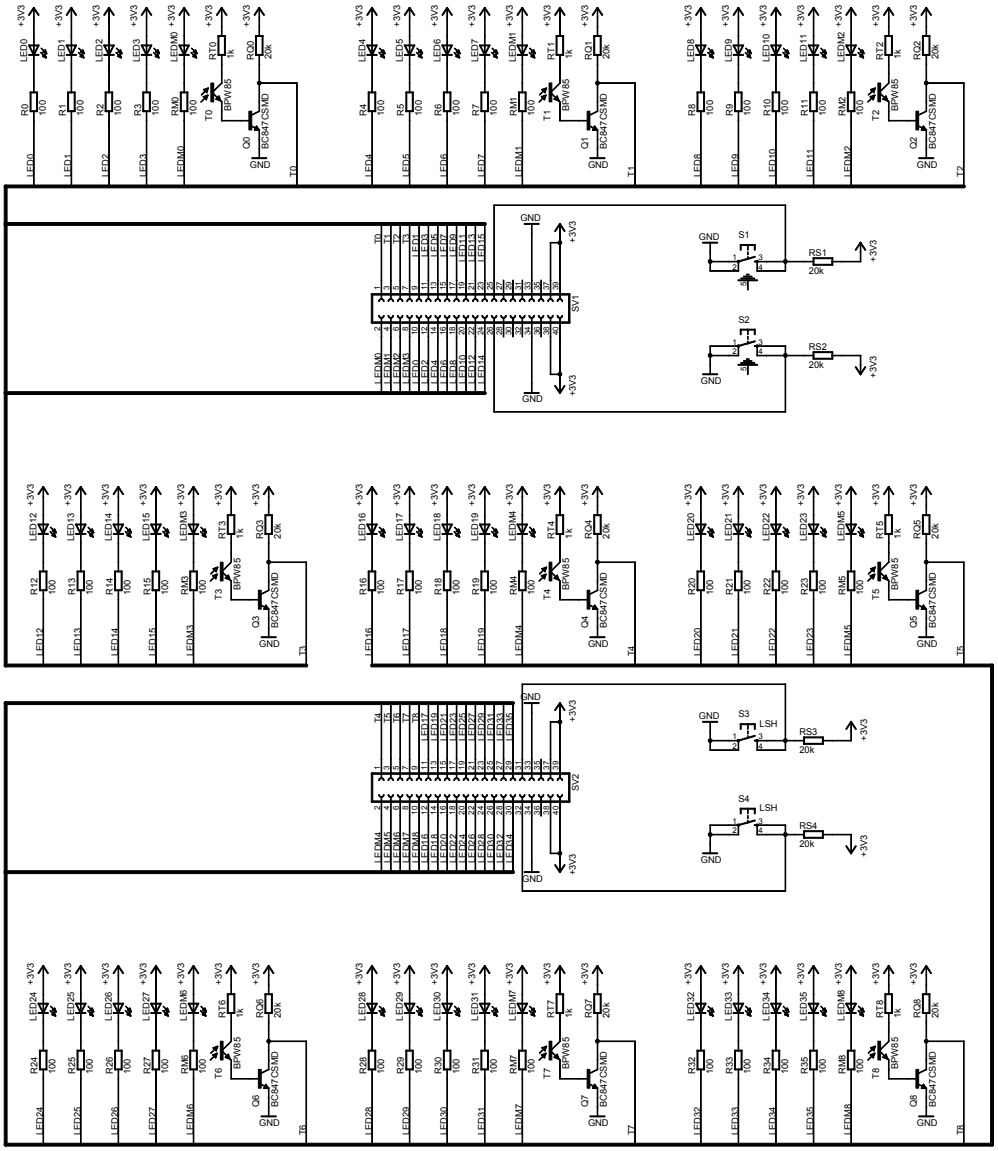


Abbildung 6.4: Schaltplan Testboard Feld

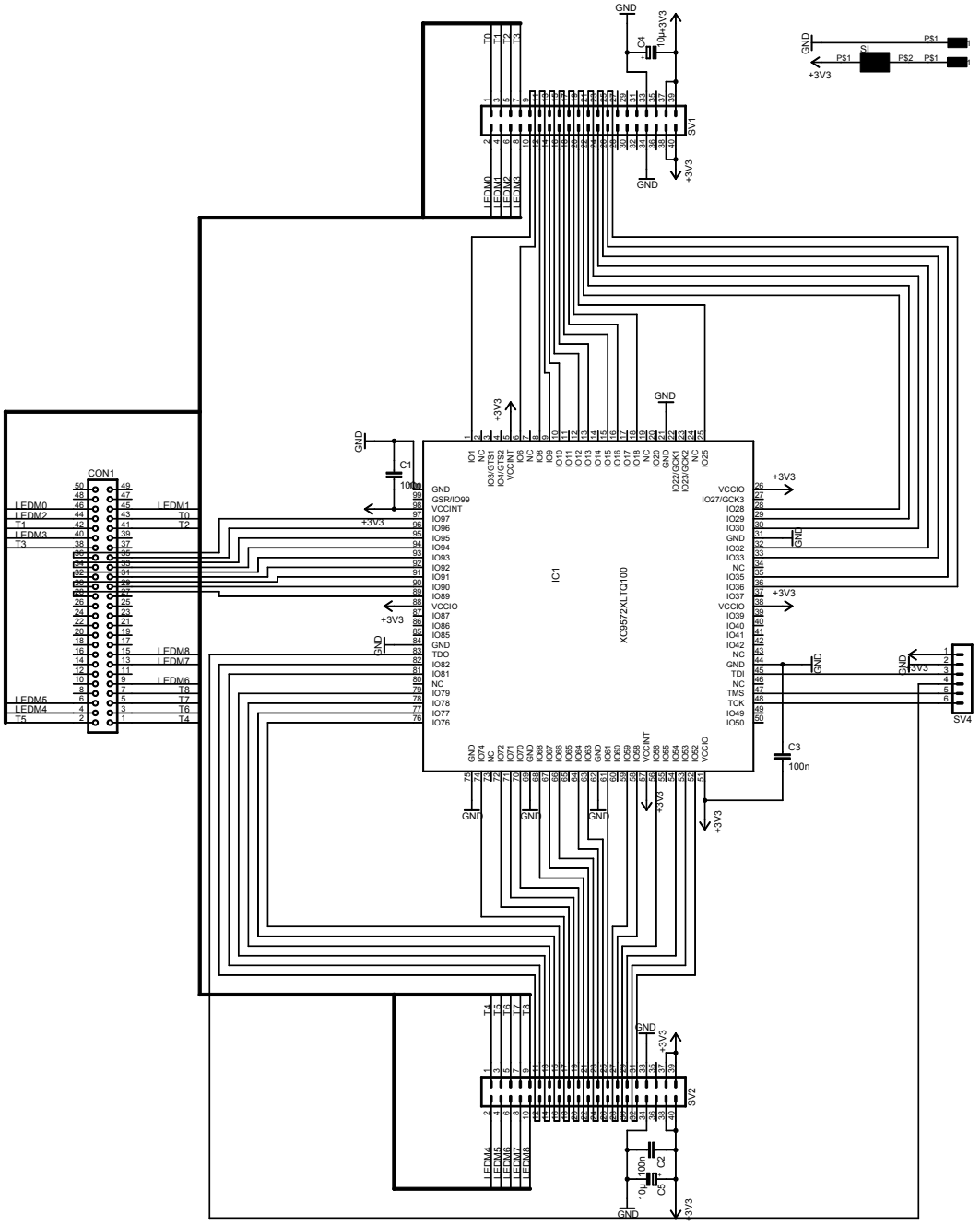


Abbildung 6.5: Schaltplan Testboard CPLD

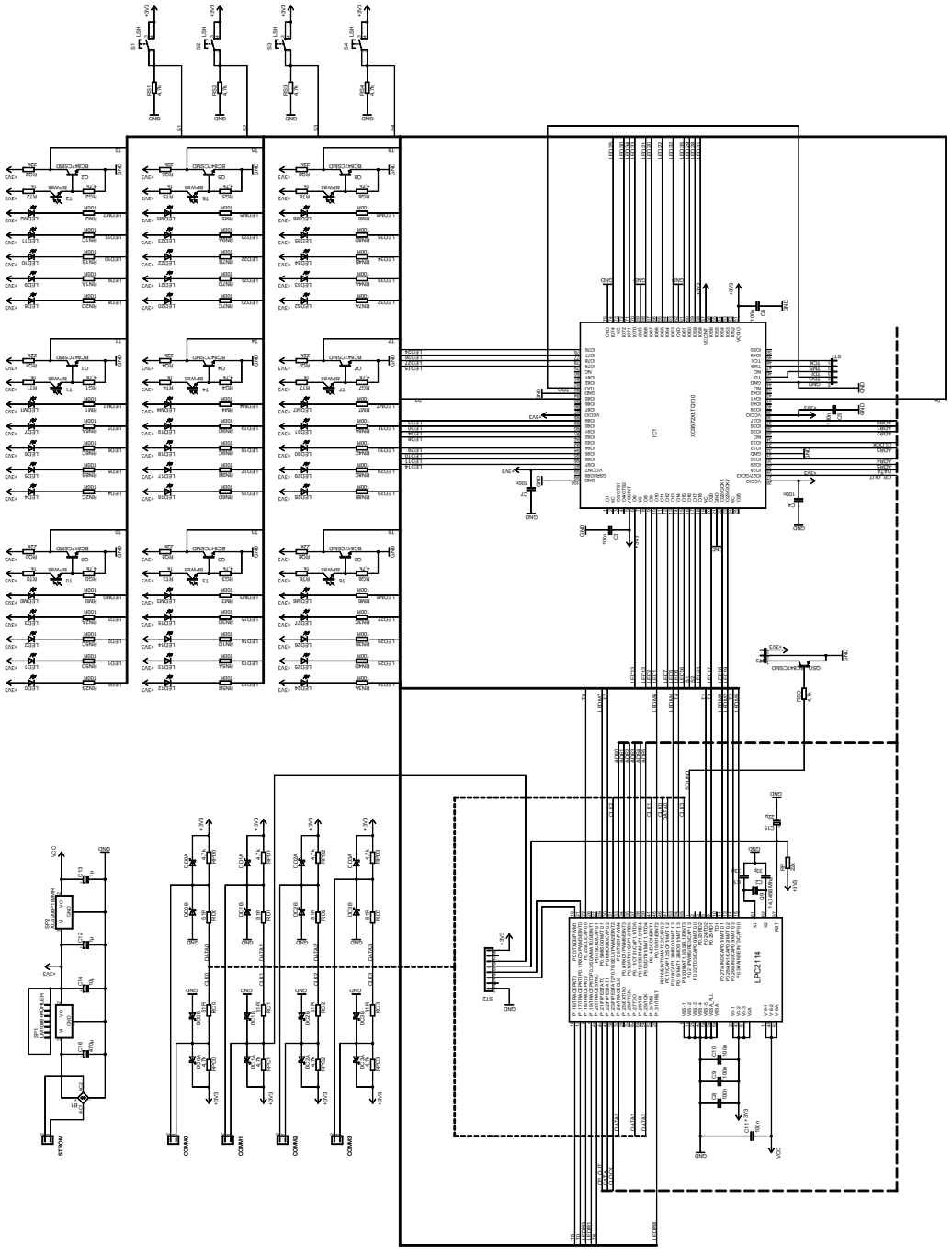


Abbildung 6.6: Schaltplan Feldplatine Version 2.0

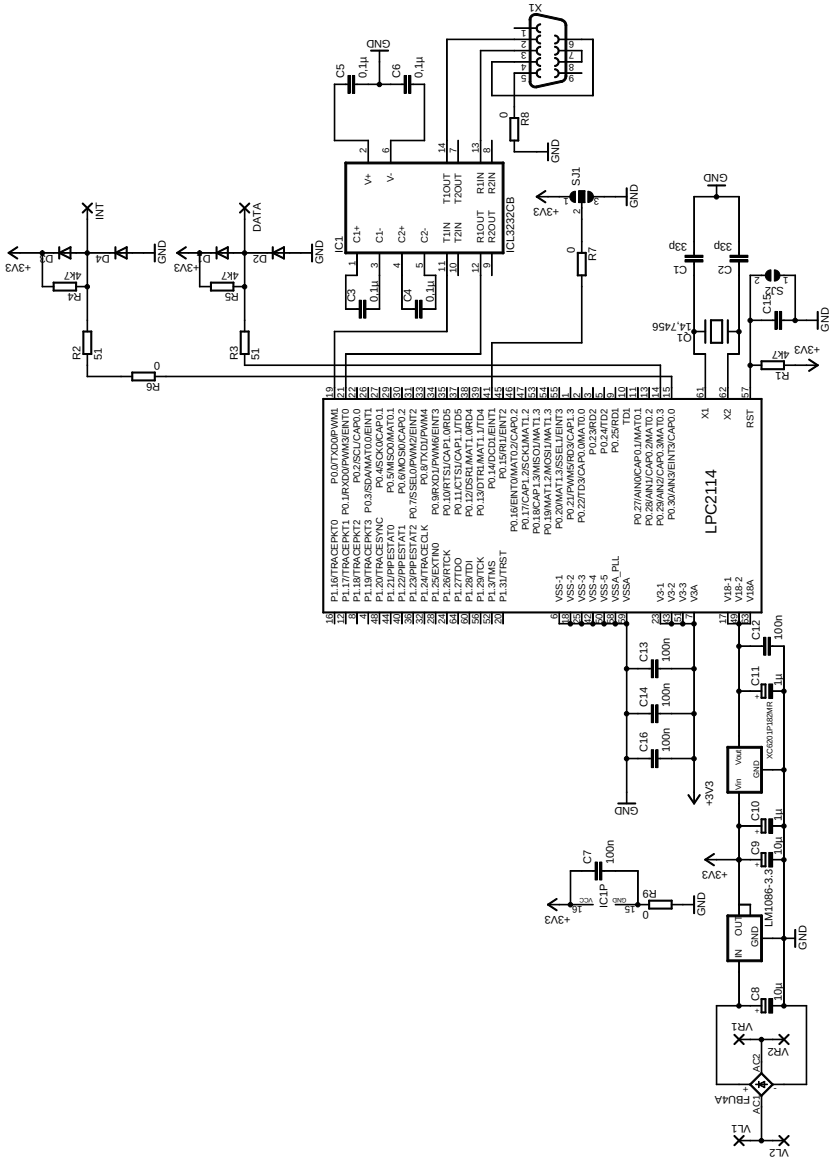


Abbildung 6.7: Schaltplan Host-Interface

6.2 Platinenlayouts

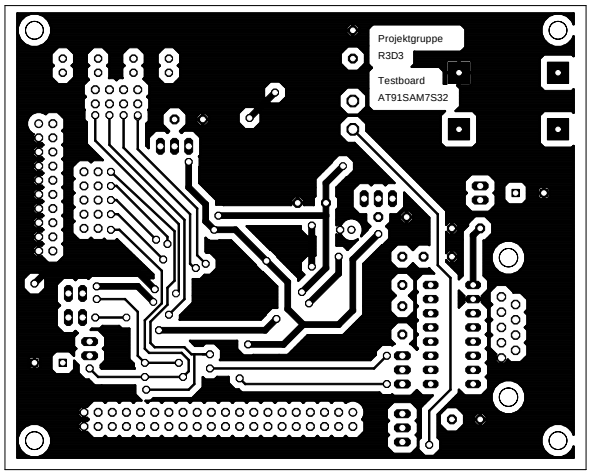


Abbildung 6.8: Platinenlayout Testboard AT91SAM7S32 - Routing - Oberseite

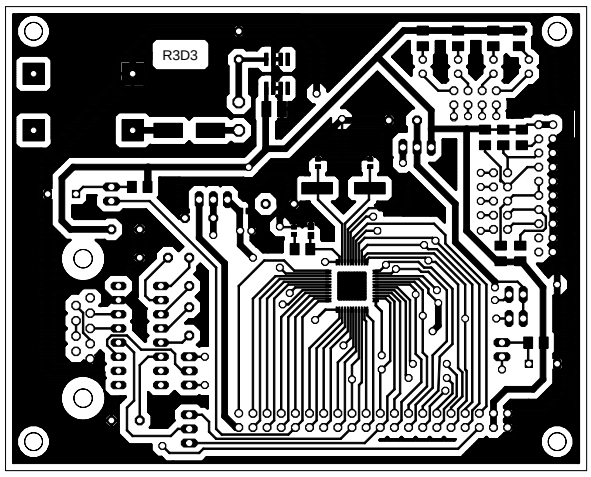


Abbildung 6.9: Platinenlayout Testboard AT91SAM7S32 - Routing - Unterseite

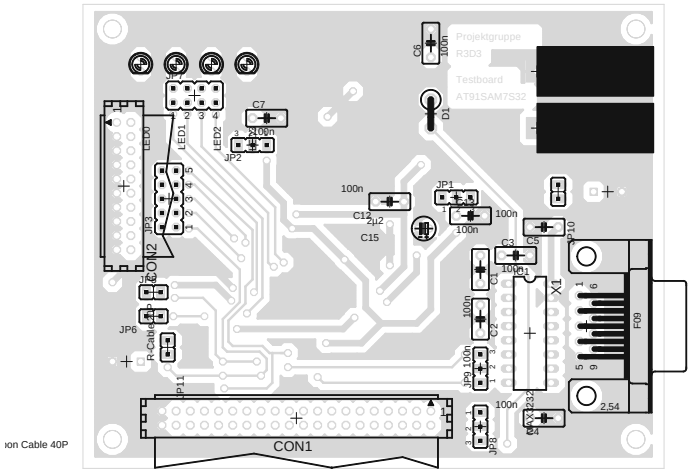


Abbildung 6.10: Platinenlayout Testboard AT91SAM7S32 - Bauteile - Oberseite

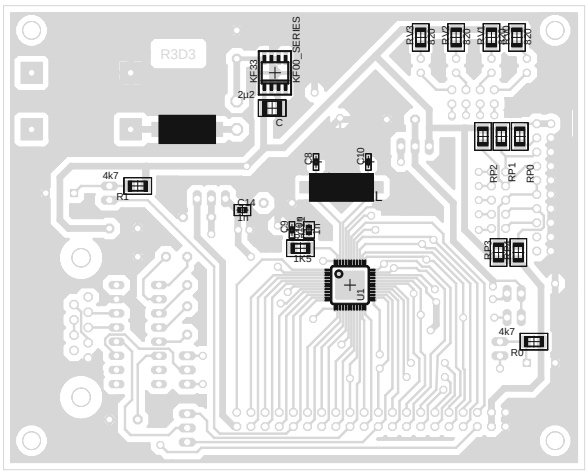


Abbildung 6.11: Platinenlayout Testboard AT91SAM7S32 - Bauteile - Unterseite

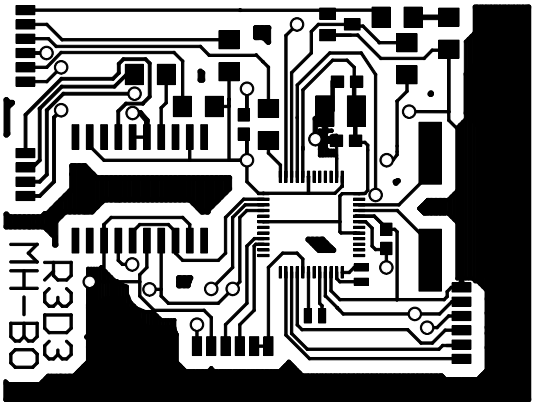


Abbildung 6.12: Platinenlayout Roboter Hauptplatine - Routing - Oberseite

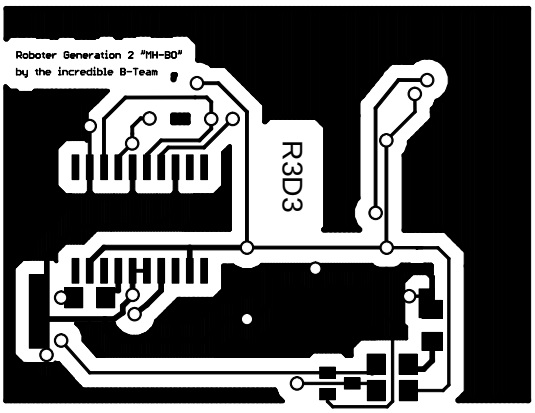


Abbildung 6.13: Platinenlayout Roboter Hauptplatine - Routing - Unterseite

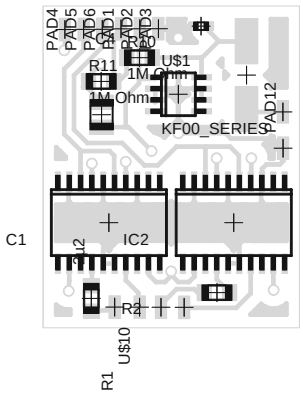


Abbildung 6.14: Platinenlayout Roboter Nebenplatine Bauteile (nur Oberseite)

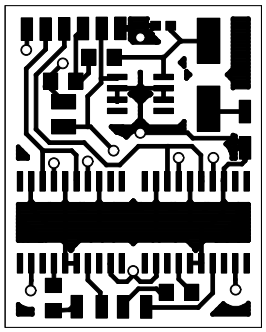


Abbildung 6.15: Platinenlayout Roboter Nebenplatine Routing Oberseite

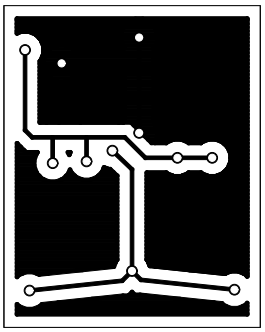


Abbildung 6.16: Platinenlayout Roboter Nebenplatine Routing Unterseite

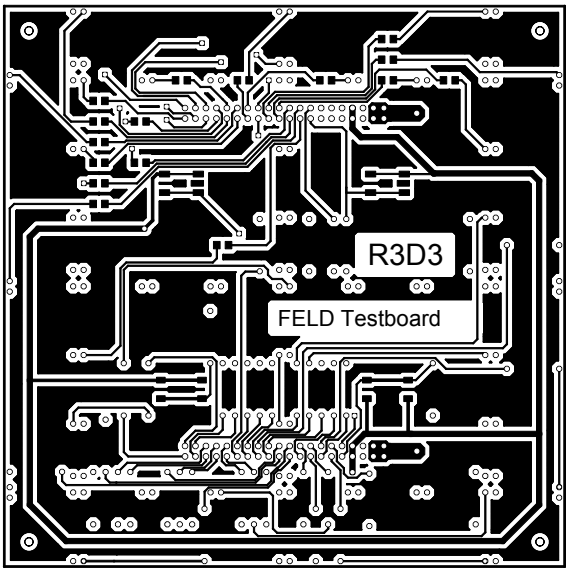


Abbildung 6.17: Platinenlayout Testboard Feld - Routing - Oberseite

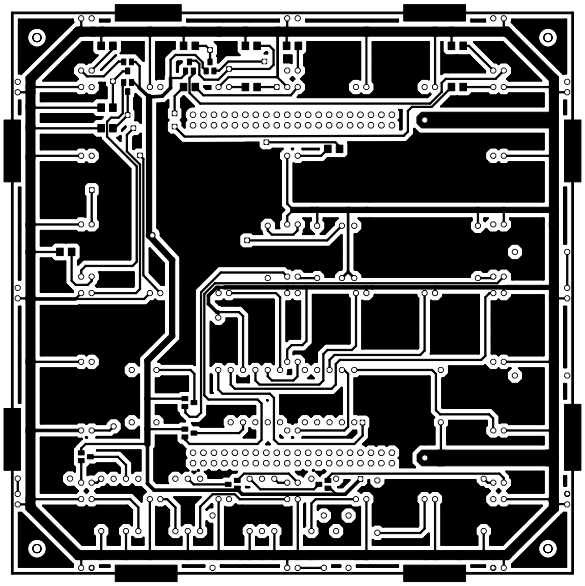


Abbildung 6.18: Platinenlayout Testboard Feld - Routing - Unterseite

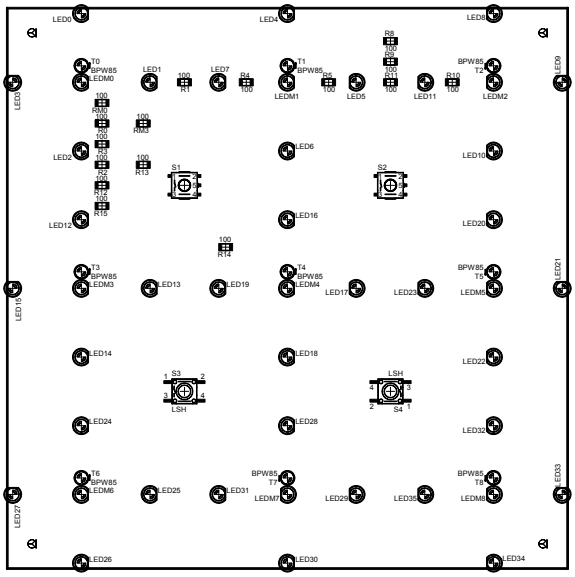


Abbildung 6.19: Platinenlayout Testboard Feld - Bauteile - Oberseite

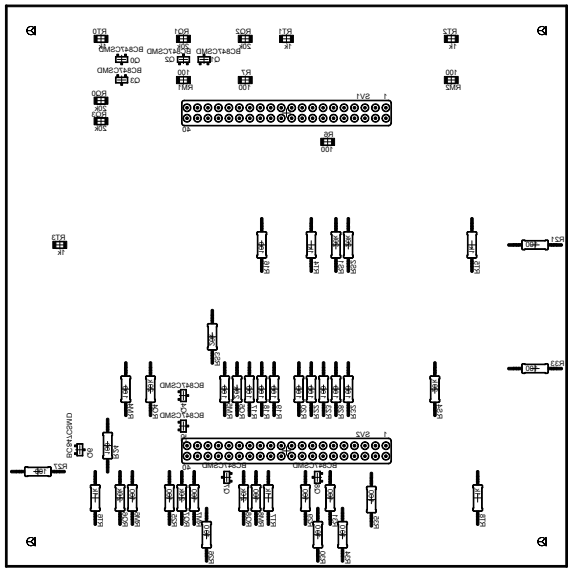


Abbildung 6.20: Platinenlayout Testboard Feld - Bauteile - Unterseite

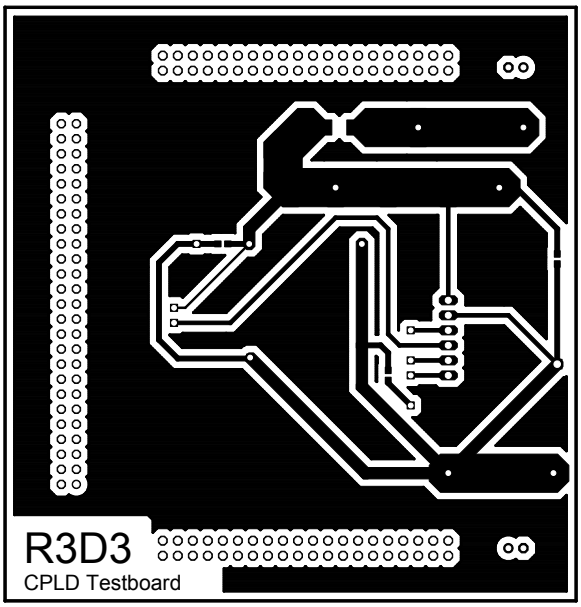


Abbildung 6.21: Platinenlayout Testboard CPLD - Routing - Oberseite

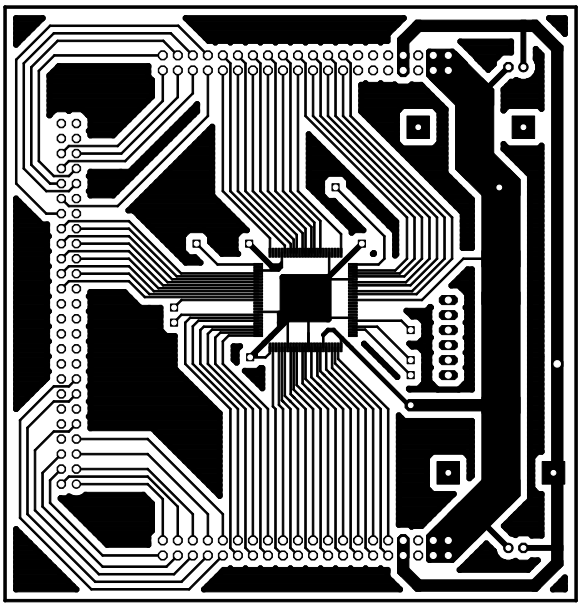


Abbildung 6.22: Platinenlayout Testboard CPLD - Routing - Unterseite

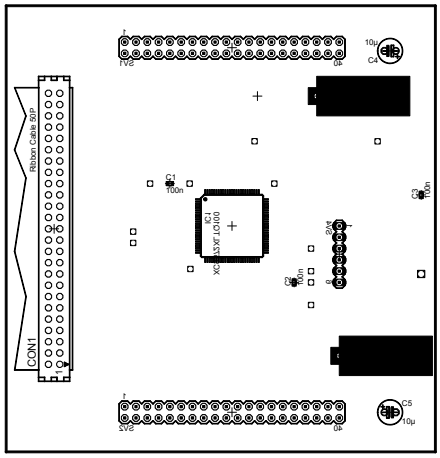


Abbildung 6.23: Platinenlayout Testboard CPLD - Bauteile - Ober- und Unterseite

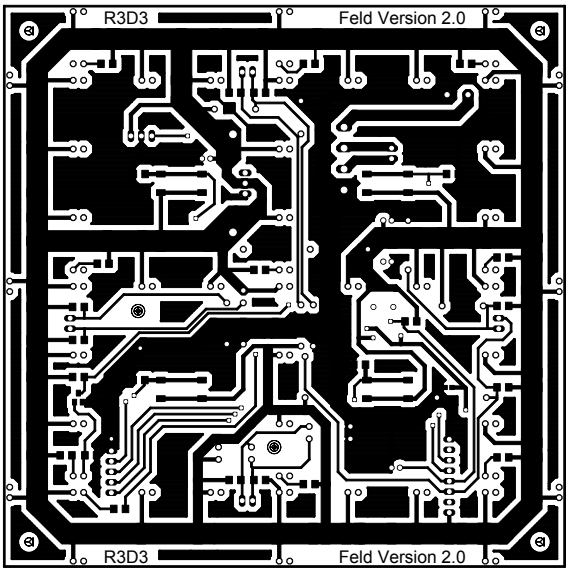


Abbildung 6.24: Platinenlayout Feldplatine Version 2.0 - Routing - Oberseite

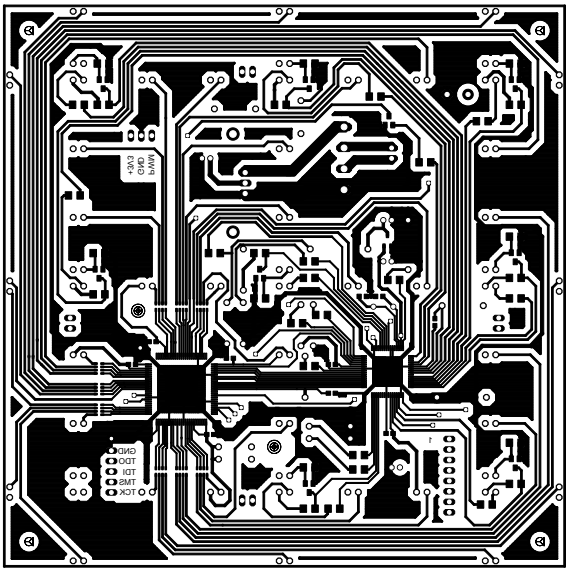


Abbildung 6.25: Platinenlayout Feldplatine Version 2.0 - Routing - Unterseite

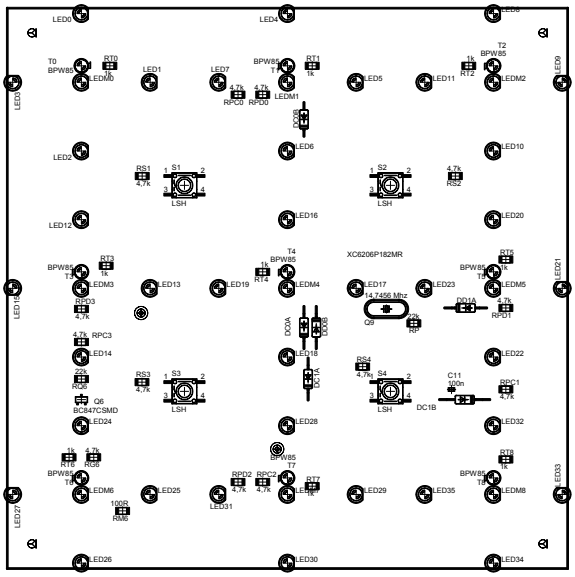


Abbildung 6.26: Platinenlayout Feldplatine Version 2.0 - Bauteile - Oberseite

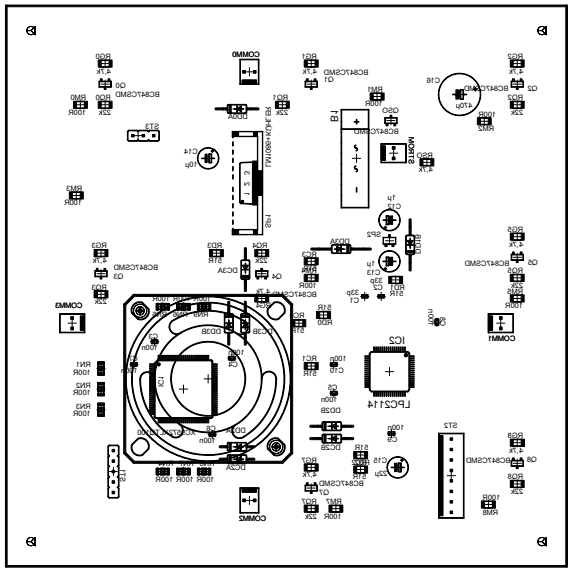


Abbildung 6.27: Platinenlayout Feldplatine Version 2.0 - Bauteile - Unterseite

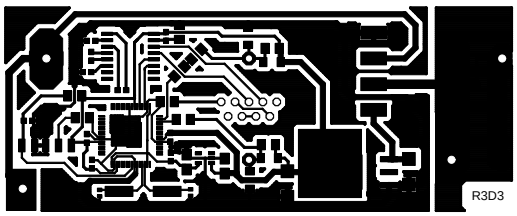


Abbildung 6.28: Platinenlayout Host-Interface - Routing - Oberseite

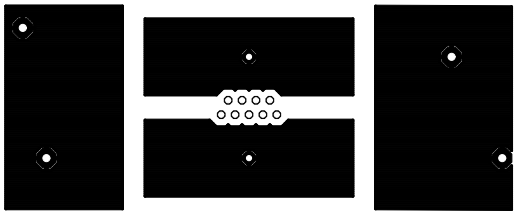


Abbildung 6.29: Platinenlayout Host-Interface - Routing - Unterseite

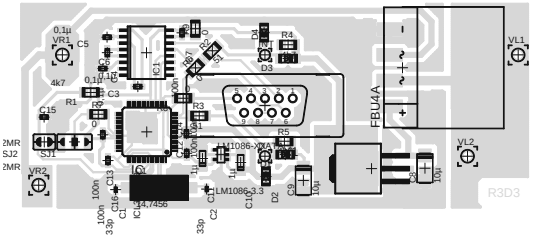


Abbildung 6.30: Platinenlayout Host-Interface - Bauteile - Oberseite

6.3 Mechanischer Aufbau des Spielfelds

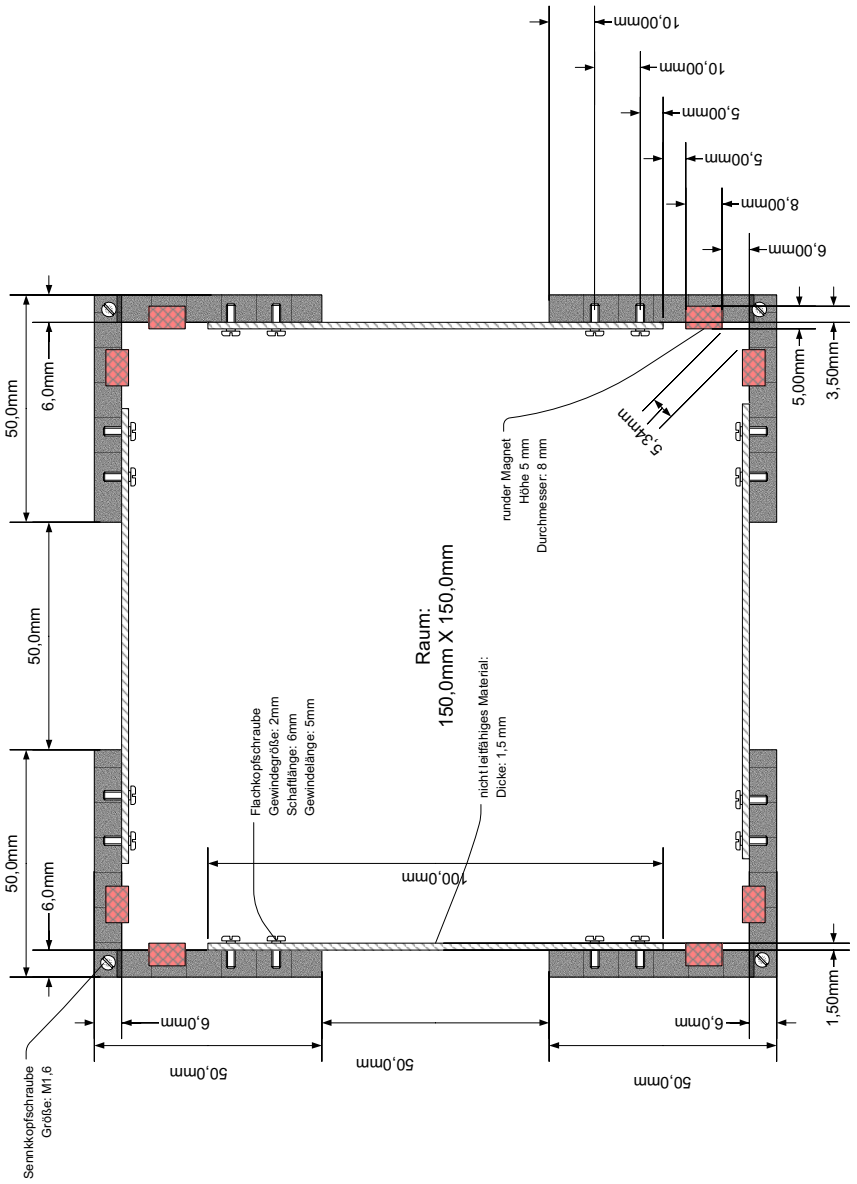


Abbildung 6.31: Spielfelfbox von oben gesehen

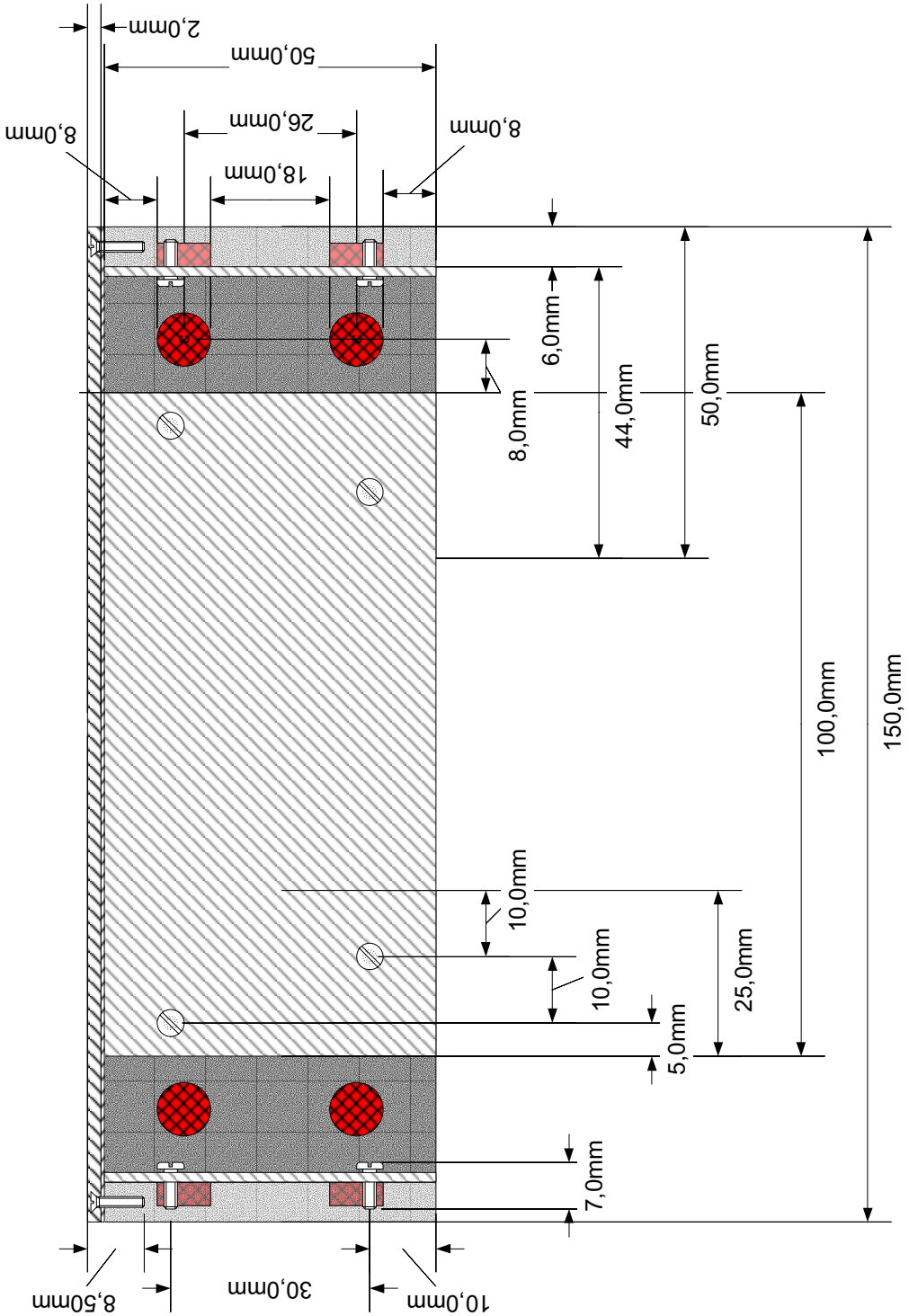


Abbildung 6.32: Spielfeldbox von der Seite gesehen

Literaturverzeichnis

[Alt98] Altenburg, Jens: *Miniaturroboter Willi.* Elekronik Laden, 1998, http://elmicro.com/files/robotik/willhand.pdf.

[AMI99] AMIGO Spiel + Freizeit GmbH: *Robo Rally - Das rasante Rennen! - Spielanleitung*, 2. Auflage, 1999, http://www.amigo-spiele.de/upload/Roborally_1091.pdf.

[AMI00] AMIGO Spiel + Freizeit GmbH: *Robo Rally Erweiterung - Crash and Burn - Spielanleitung*, 1. Auflage, 2000, http://www.amigo-spiele.de/upload/RoboRally2_1092.pdf.

[ARM05] ARM Ltd: *ARM7TDMI - ARM 32-bit RISC core with 16-bit system costs*, Oct 2005, http://www.arm.com/products/CPUs/ARM7TDMI.html.

[Atm03] Atmel Corporation: *Atmel AT91RM3400 Datasheet Rev. 1790A*, Nov 2003, http://www.atmel.com/dyn/resources/prod_documents/doc1790.pdf.

[Atm05a] Atmel Corporation: *Atmel AT91M42800A Datasheet Rev. 1779C*, Nov 2005, http://www.atmel.com/dyn/resources/prod_documents/doc1779.pdf.

[Atm05b] Atmel Corporation: *Atmel AT91M55800A Datasheet Rev. 1745G*, Nov 2005, http://www.atmel.com/dyn/resources/prod_documents/doc1745.pdf.

[Atm05c] Atmel Corporation: *Atmel AT91SAM7S32 Datasheet Rev. 6175A*, Jul 2005, http://www.atmel.com/dyn/resources/prod_documents/doc6175.pdf.

[Atm05d] Atmel Corporation: *Atmel ATmega162 Datasheet Rev. 2513G*, Mar 2005, http://www.atmel.com/dyn/resources/prod_documents/doc2513.pdf.

[Atm05e] Atmel Corporation: *Atmel ATmega6490 Datasheet Rev. 2552B*, May 2005, http://www.atmel.com/dyn/resources/prod_documents/doc2552.pdf.

[Atm05f] Atmel Corporation: *Atmel ATtiny26L Datasheet Rev. 1477G*, Mar 2005, http://www.atmel.com/dyn/resources/prod_documents/doc1477.pdf.

[Atm05g] Atmel Corporation: *Atmel AVR 8-Bit RISC Summary*, Oct 2005, http://www.atmel.com/products/AVR/overview/overview7.asp.

[Atm06a] Atmel Corporation: *Atmel AT90PWM2 Datasheet Rev. 4317E*, Feb 2006, http://www.atmel.com/dyn/resources/prod_documents/doc4317.pdf.

[Atm06b] Atmel Corporation: *Atmel AT91SAM7S32 Flash Tool*, Feb 2006, http://www.at91.com/Pages/products/microcontroller/AT91SAM/AT91SAM7S/at91sam7s.html.

[B+06a] Brall, Frank and others: *Bild:Schrittmotorprinzip2.gif*. Webpage, February 2006, http://www.roboternetz.de/wissen/index.php/Bild:Schrittmotorprinzip2.gif.

[B+06b] Brall, Frank and others: *Getriebemotoren*, February 2006, http://www.roboternetz.de/wissen/index.php/Getriebemotoren.

[B+06c] Brall, Frank and others: *Roboter-Typen*, January 2006, http://www.roboternetz.de/wissen/index.php/Roboter-Typen.

[Bac03] Back, Wolfgang: *Der Schrittmotor*, 2003, http://www.wolfgang-back.com/PDF/Schrittmotor.pdf.

[BSW06] Bachfeld, Daniel, Thomas J. Schult, and Carsten Wiese: *Auf Asimovs Spuren*. c't, 2:136–143, January 2006.

[BTT06] Benz, Benjamin, Carl Thiede, and Thorsten Thiele: *Spielgefährten*. c't, 2:130–135, January 2006.

[CS05] Cisco Systems, Inc.: *OSPF Design Guide Document ID: 7039*, August 2005, http://www.cisco.com/warp/public/104/1.html.

[Dio05] Diotec: *KBU4A...KBU4M*, October 2005, http://www.diotec.com/pdf/kbu4.pdf.

[DPR] *Brief H-Bridge Theory of Operation*. Webpage, http://www.dprg.org/tutorials/1998-04a/index.html.

[Fau] Faulhaber Group: *Antriebssysteme*, Katalog 2003/2004.

[Fra06] *Roboternetz - Roboter, Elekronik, Mikrocontroller*, Januar 2006, http://www.roboternetz.de.

[GG04] Grewe, Jan und Robin Gruber: *ASURO Roboterbausatz - Modell ARX-03*. DLR and AREXX, 2004, http://www.schoollab.dlr.de/Schoollab/Oberpfaffenhofen/Aktuelles/191164-an-01-de-Asuro_BS_Programmierbarer_Roboter.pdf.

[Gra04] Graupner GmbH & Co KG: *Bedienungsanleitung ULTRAMAT 12*, November 2004,

http://www.graupner.de/fileadmin/downloadcenter/anleitungen/6412.69.pdf.

[HER06] HERPA MINIATURMODELLE GMBH: *LKW Räder Produktbeschreibung*, 2006, http://www.herpa.de/collect/(54hxeo55fly22l55tvxtcgui)/detail.aspx?ProductID=051040&lang=de-DE&express=&thumb=1.

[Hit] *Robonova-I*. Webshop, http://www.robonova.de/store/product.php?productid=16135&cat=248&page=1.

[Int02] INTERSIL CORPORATION: *ICL3221, ICL3222, ICL3223, ICL3232, ICL3241, ICL3243*, March 2002, http://www.intersil.com/data/fn/fn4805.pdf.

[Int05] INTERSIL: *Half Amp Full Bridge Power Driver for Small 3V, 5V and 12V DC Motors*, December 2005, http://www.intersil.com/data/fn/fn3976.pdf.

[Kei] *Keil uVision*, http://www.keil.com/uvision2/.

[Kei05] KEIL - AN ARM COMPANY: *Keil: ARM Development Tools*, Oct 2005, http://www.keil.com/arm/.

[KTe] *Robot Khepera II*. Webpage, http://www.k-team.com/kteam/index.php?site=1&rub=22&page=17&version=EN.

[LCP+04] LUA, ENG KEONG, JON CROWCROFT, MARCELO PIAS, RAVI SHARMA, and STEVEN LIM: *A survey and comparison of peer-to-peer overlay network schemes*. IEEE COMMUNICATIONS SURVEY AND TUTORIAL, MARCH 2004.

[Lyn] *Extreme Hex 3*. Webshop, http://www.lynxmotion.com/Category.aspx?CategoryID=33.

[Max03] MAXIM: $3.0V$ *to* $5.5V$*, Low-Power, up to* $1Mbps$*, True RS-232 Transceivers Using Four* $0,1\mu F$ *External Capacitors*, 2003, http://pdfserv.maxim-ic.com/en/ds/MAX3222-MAX3241.pdf.

[McM04] MCMANIS, CHUCK: *Voltage Spikes in FET based H-bridges*. Webpage, 2004, http://www.mcmanis.com/chuck/robotics/projects/esc2/hbridge_spiking.html.

[Mik03a] MIKROANTRIEBE: *Getriebemotor DC Bausatz (zum Kleben)*, Juli 2003, http://www.mikroantriebe.de/download/g30bs_a.pdf.

[Mik03b] MIKROANTRIEBE: *Mikromotor M61513*, Januar 2003, `http://www.mikroantriebe.de/download/m660m665.pdf`.

[Mik05] *Technik für funkferngesteuerte Mikromodelle*, August 2005, Produktübersicht und Preise 08/2005, `http://www.mikroantriebe.de/download/preisliste.pdf`.

[Nan] *NanoTec Antriebstechnik*, Katalog 2001, `http://www.lynxmotion.com/Category.aspx?CategoryID=33`.

[Nan04a] NANOTEC: *Stepping Motor - SP0618M0104-A-Z01*, July 2004, `http://www.nanotec.de/data/zweiphasen/sp1555/media/sp0618m0104.pdf`.

[Nan04b] NANOTEC: *Stepping Motor - SP1018M0204-A-Z01*, July 2004, `http://www.nanotec.de/data/zweiphasen/sp1555/media/sp1018m0204.pdf`.

[Nat] NATIONAL SEMICONDUCTORS: *LMD18200 – 3A, 55V H-Bridge*.

[Nat05] NATIONAL SEMICONDUCTOR: *LM1086* 1.5*A Low Dropout Positive Regulators*, June 2005, `http://cache.national.com/ds/LM/LM1086.pdf`.

[Par] *Boe-Bot Robot Kit - Serial Version*. Webshop, `http://www.parallax.com/detail.asp?product_id=28132`.

[Phi03] PHILIPS SEMICONDUCTORS: *LPC2114/2124/2212/2214 USER MANUAL*, May 2003, `http://www.semiconductors.philips.com/acrobat/usermanauls/UM_LPC2114_2124_2212_2214_2.pdf`.

[Poe36] POE, EDGAR ALLAN: *Maelzel's Chess-Player*. Southern Literary Journal, April 1836.

[Pri] PRINZING, FABIAN: *Motortreiberschaltung mit 74245 TTL-Logik 8-Bus-Transceiver*. Webpage, `http://www.the-starbearer.de/Schaltungen/Mototreiber74245.htm`.

[Red04] REDAKTIONSBEITRAG: *Deutsche Roboter holen Fussball-WM-Titel*. Netzeitung, Juli 2004.

[SAN] SANYO: *saturation, Bidirectional Motor Driver for Low-voltage Applications*, `http://www.ortodoxism.ro/datasheets/sanyo/ds_pdf_e/LB1836M.pdf`.

[Sch87] SCHWARTZ, MISCHA: *Telecommunication Networks, Protocols, Modelling and Analysis*. Addison Wesley, 1987.

[Sma05] Smart, Julian: *Cross-Platform GUI Programming with wxWidgets*. Prentice Hall Professional Technical Reference, 2005.

[Son] *Sony AIBO Europe*. Webpage, http://www.aibo-europe.com/index.asp?language=de.

[STM00] STMicroelectronics: *DUAL FULL-BRIDGE DRIVER*, January 2000, http://www.st.com/stonline/products/literature/ds/1773.pdf.

[STM04] STMicroelectronics: *KF00 SERIES*, 2004, http://www.st.com/stonline/products/literature/ds/4337/kf33b.pdf.

[Tel96] TelCom Semiconductors: *LOGIC-INPUT CMOS QUAD DRIVERS (TC4467, TC4468, TC4469)*, October 1996, http://www.ortodoxism.ro/datasheets/TelComSemiconductor/mXtzyqt.pdf.

[Tex04] Texas Instruments: *L293, L293D - Quadruple half -H drivers - SLRS008C*, November 2004, http://focus.ti.com/lit/ds/symlink/l293d.pdf.

[Tora] Torex: *XC6201 Series Positiv Voltage Regulators*, http://www.torex-usa.com/product/pro02/pdf/XC6201_E.pdf.

[Torb] Torex: *XC6206 Series Low ESR Cap. Compatible Positiv Voltage Regulators*, http://www.torex-usa.com/product/pro02/pdf/XC6206.pdf.

[vV] Vliet, Bas van: *Freeforming the H-bridge*. Webpage, http://www.geocities.com/beamnet/H-bridge.htm.

[Wes04] *Katalog-D*, April 2004, http://www.wes-technik.de/Download/Katalog-D.pdf.

[Wiz05] Wizards of the Coast, Inc.: *Robo Rally - Rulebook*, 2005, http://www.wizards.com/avalonhill/rules/roborally.pdf.

[Xil98a] Xilinx Inc.: *JTAG Parallel Download Cable Schematic*, November 1998, http://toolbox.xilinx.com/docsan/data/alliance/jtg/fig26.htm.

[Xil98b] Xilinx Inc.: *Planning for High Speed XC9500XL Designs*, September 1998, http://www.xilinx.com/bvdocs/appnotes/xapp115.pdf.

[Xil99a] Xilinx Inc.: *Designing With XC9500XL CPLDs*, January 1999, http://www.xilinx.com/bvdocs/appnotes/xapp112.pdf.

[Xil99b] Xilinx Inc.: *JTAG Programmer Guide*, July 1999, http://www.xilinx.com/support/sw_manuals/2_1i/download/jtag.pdf.

[Xil02a] XILINX INC.: *A Quick JTAG ISP Checklist*, June 2002, `http://www.xilinx.com/bvdocs/appnotes/xapp104.pdf`.

[Xil02b] XILINX INC.: *Using the XC9500/XL/XV JTAG Boundary Scan Interface*, December 2002, `http://www.xilinx.com/bvdocs/appnotes/xapp069.pdf`.

[Xil03] XILINX INC.: *Xilinx XC9500 CPLD Series*, November 2003, `http://www.xilinx.com/products/95xlsh.pdf`.

[Xil05a] XILINX INC.: *Bulletproof CPLD Design Practices*, June 2005, `http://www.xilinx.com/bvdocs/appnotes/xapp784.pdf`.

[Xil05b] XILINX INC.: *Driving LEDs With Xilinx CPLDs*, April 2005, `http://www.xilinx.com/bvdocs/appnotes/xapp805.pdf`.

[Xil05c] XILINX INC.: *XC9572XL High Performance CPLD*, July 2005, `http://www.xilinx.com/bvdocs/publications/ds057.pdf`.

[Zet04] ZETEX SEMICONDUCTORS: *Complementary 60 V enhancement mode MOSFET H-bridge*, March 2004, `http://www.ortodoxism.ro/datasheets/zetexsemiconductors/zxmhc6a07t8.pdf`.